MICRO-ORDINATEURS 

# LA CONDUITE DU ZX 81

Gabriel NOLLET

ZX81

EYROLLES

sylvine bouvry

# LA CONDUITE DU ZX 81

DANS LA MÊME COLLECTION

SCHOMBERG — Le Basic universel.
SCHOMBERG — Micro-ordinateurs: comment ça marche?
HERNANDEZ — Pascal par l'exemple.
NOLLET — La conduite du ZX 81.
PELLIER — La conduite du TRS 80.
LADEVIE — Votre gestion avec BASIC sur micro-ordinateur.
QUEINNEC — Langage d'un autre type: LISP.
PELLIER — Programmez vos jeux d'action rapide sur TRS 80.
ASTIER — La conduite de L'APPLE II
Tome 1: le Basic de l'APPLE II
Tome 2: le système graphique et l'assembleur de l'APPLE II.
MONTEIL — L'assembleur facile du 6502.
LEPAPE — L'assembleur facile du Z 80.
OROS et — Zx 81 à la conquête des jeux
PERBOST — CASSETTE — ZX 81 à la conquête des jeux.
DAX — CP/M et sa famille. Guide d'utilisation.
NOLLET — Langage machine, trucs et astuces sur ZX 81.
BICKING — La conduite du PC 1211 (ou TRS 80 pocket).
TEJA — Apprenez à parler à votre ordinateur.
MONTEIL — La conduite du VIC 20.
SAGUEZ — Contrôle. Commande par micro-ordinateur.



# LA CONDUITE DU ZX 81

par

**Gabriel NOLLET**

Collection animée
par Richard SCHOMBERG

DEUXIÈME ÉDITION
nouveau tirage



61, Boulevard Saint-Germain — 75005 Paris
1983

Si vous désirez être tenu au courant de nos publications, il vous suffit d'adresser votre carte de visite au :

Service «Presse», Éditions EYROLLES
61, Boulevard Saint-Germain,
75240 PARIS CEDEX 05,

en précisant les domaines qui vous intéressent. Vous recevrez régulièrement un avis de parution des nouveautés en vente chez votre libraire habituel.

# Avant-propos

*Avec le ZX 80, SINCLAIR nous avait déjà étonnés. Avec le ZX 81 destiné à prendre la relève de son illustre prédécesseur, nous assistons littéralement à une révolution dans le monde de l'ordinateur domestique. Son prix attractif et ses qualités ont trouvé auprès d'un large public un écho favorable, et le nombre de ses acquéreurs augmente de jour en jour.*

*LA CONDUITE DU ZX 81 s'adresse à tous ceux qui possèdent déjà une connaissance du langage BASIC (ou de tout autre langage) et qui utilisent ou envisagent d'utiliser un SINCLAIR ZX 81.*

*Ceux qui, abordant pour la première fois l'informatique, désirent apprendre le langage BASIC ou se familiariser avec le fonctionnement interne des ordinateurs, liront avec profit le BASIC UNIVERSEL et MICRO-ORDINATEURS : COMMENT ÇA MARCHE ? publiés dans la même collection. La lecture de ces deux livres de base vous permettra d'aborder sans aucune difficulté le manuel BASIC de n'importe quel micro-ordinateur.*

*Après avoir lu LA CONDUITE DU ZX 81, vous saurez comment avoir accès à la programmation en langage machine, pourquoi il vaut mieux écrire LET A = C = C plutôt que LET A = 1, la manière de réaliser des graphiques animés. Ceux qui possèdent un ZX 80 n'ont cependant pas été délaissés ! Ils apprendront comment faire pour utiliser la commande SLOW comme sur le ZX 81 ! Vous trouverez aussi la description de quelques-unes des extensions (mémoire, clavier, ports d'entrée-sortie...) que vous pourrez vous procurer aisément, faisant de votre ZX 81 (ou de votre ZX 80 adapté) un outil extrêmement sophistiqué et performant.*

# Table des matières

**Photos :** M. RAMPNOUX



# Introduction

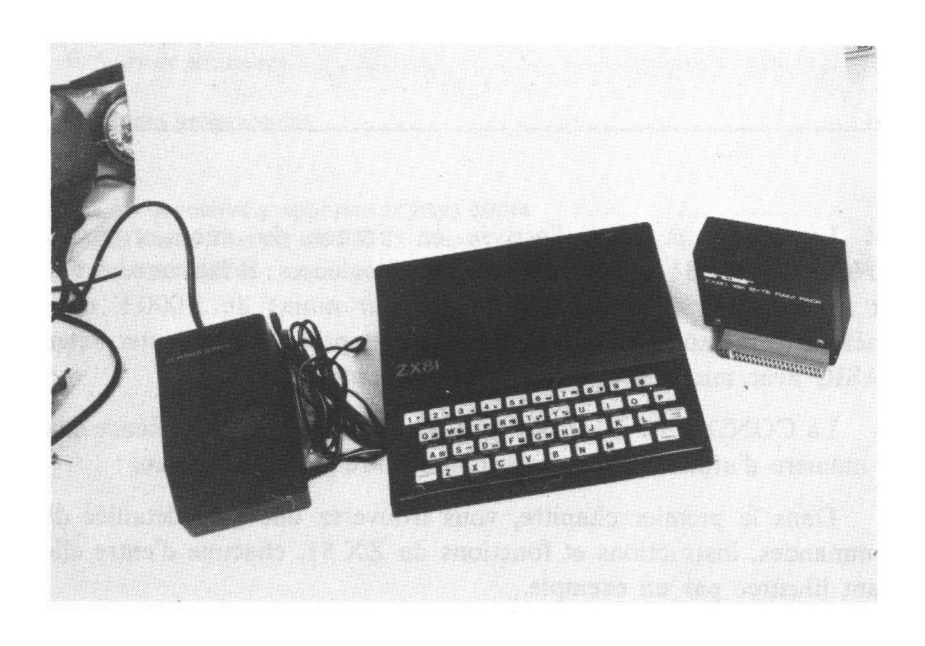
La presse a salué l'arrivée en France du micro-ordinateur SINCLAIR ZX 81 par des critiques fort élogieuses : il faut avouer qu'il est remarquable de pouvoir disposer pour moins de 1 000 F d'une machine à mémoire vive de 1 K-octets, et possédant un interpréteur BASIC avec une arithmétique virgule flottante !

La CONDUITE DU ZX 81 vous apportera une aide efficace dans la manière d'utiliser et de programmer votre micro-ordinateur :

Dans le premier chapitre, vous trouverez une liste détaillée des commandes, instructions et fonctions du ZX 81, chacune d'entre elles étant illustrée par un exemple.

- Le chapitre 2 vous donnera quelques conseils pour économiser la place en mémoire, et vous indiquera comment chaîner des programmes entre eux.
- Le chapitre 3 traite de la programmation en langage machine.
- Le graphique, base de tous les jeux, est abordé au chapitre 4.
- Le chapitre 5 s'adresse plus particulièrement aux possesseurs de ZX 80 ! Ils y découvriront comment le "transformer" en ZX 81 !

Les autres chapitres vous donneront plusieurs petits "trucs" pour améliorer encore les performances de votre ZX 81.




*Alimentation + ZX 81 + bloc extension 16 K RAM*

# 1

# Les instructions BASIC du ZX 81

Ce chapitre s'adresse surtout à ceux qui pratiquent déjà le langage BASIC et qui désirent connaître toutes les subtilités du BASIC du ZX 81. Chacune des instructions et commandes du BASIC y est reprise, expliquée, et accompagnée d'un exemple.

Toutes ces instructions ont été classées par ordre alphabétique de façon à pouvoir les retrouver facilement. N'hésitez pas à relire la description d'une instruction particulière si vous avez un doute au sujet de sa syntaxe.

## 1.1. LES COMMANDES DU BASIC DU ZX 81

Dans ce paragraphe nous étudions les commandes du BASIC du ZX 81. Certaines de ces commandes peuvent être utilisées comme des instructions. Vous les retrouverez donc dans le paragraphe "Instructions". D'autre part, toutes les instructions peuvent être utilisées comme des commandes ! Il suffit pour cela de rentrer l'instruction sans mettre de numéro de ligne.

*Exemple :*

```
PRINT A
```

**CONT** : Commande.

CONT

La commande CONT permet de reprendre l'exécution d'un programme à partir de l'instruction à laquelle il s'est arrêté. (Arrêt dû à l'exécution d'une instruction STOP ou à un appui sur la touche BREAK ou à une erreur de programme). Si l'arrêt a été provoqué par une instruction STOP, l'exécution recommencera à partir de l'instruction suivante.

*Exemple :*

Le petit programme suivant permettra de calculer les dix premières coordonnées des points d'une courbe.

```
 10  PRINT  "ENTREZ L EQUATION DE VOTRE COURBE AINSI"
 20  PRINT  "70 LET Y = 3X + 4"
 30  PRINT  "PUIS APPUYER SUR NEWLINE"
 40  PRINT  "ENSUITE APPUYER SUR CONT ET NEWLINE"
 50  STOP
 60  FOR    I = 1 TO 10
 70  LET    Y = 0
 80  PRINT  "Y =" ; Y ;" POUR X =" ; I
 90  NEXT   I
100  STOP
```

L'exécution du programme sera arrêtée une première fois par l'instruction 50 STOP. Après avoir entré l'équation de la manière indiquée, le fait d'appuyer sur CONT fera repartir l'exécution du programme à partir de l'instruction 60 FOR I = 1 TO 10.

**EDIT** : Commande.

EDIT

La commande EDIT recopie en bas de l'écran la ligne pointée par le curseur. La ligne recopiée peut alors être modifiée. Pour pouvoir corriger une instruction située au milieu de votre programme, faites LIST suivi du numéro de ligne de cette instruction. Le curseur pointera sur cette instruction et il suffira alors d'entrer la commande EDIT pour la modifier.



**FAST** : Commande.

Cette commande met le ZX 81 dans le mode de travail "rapide". Dans ce mode de travail votre ZX 81 donne la priorité au calcul et n'affiche les résultats que dans les cas suivants :

— exécution d'une instruction PAUSE ;
— exécution d'une instruction INPUT ;
— exécution d'une instruction STOP.

Le microprocesseur de votre ZX 81 doit gérer le clavier (détecter si vous avez appuyé sur une touche et dans ce cas la reconnaître), l'affichage (envoyer les caractères à afficher à votre télévision) et exécuter votre programme. Cependant, il ne peut faire qu'une seule de ces choses à la fois. Ainsi lorsqu'il calcule ou lorsque vous appuyez sur une touche, il lui est matériellement impossible de s'occuper de l'affichage, ce qui provoque la disparition momentanée de celui-ci. (Le ZX 80 ne possède que ce mode de fontionnement.)

*Voir aussi instruction SLOW et PARAGRAPHE GRAPHIQUES ANIMÉS du chapitre POSSIBILITÉS GRAPHIQUES.*

**GOTO** : Commande.

GOTO n
n numéro d'une instruction de votre programme.

Cette forme de l'instruction GOTO utilisée comme une commande a le même effet que la commande RUN n, avec la différence suivante : GOTO ne remet pas les valeurs des variables à zéro au début de l'exécution de votre programme contrairement à l'instruction RUN.

*Voir aussi instruction RUN.*

**LIST** : Commande.

LIST n
n numéro d'une instruction de votre programme.

La commande LIST n permet d'afficher le listing du programme contenu en mémoire en commençant à l'instruction ayant le numéro n. Si

la totalité du programme ne peut tenir sur l'écran, vous en serez avertis par un code de type 5.

La commande LIST n, puis EDIT recopie la ligne de programme n dans le bas de l'écran, autorisant une modification rapide de cette ligne de programme.

**LOAD**: Commande.

LOAD"n"

n est le nom donné au programme dans la commande SAVE lors de la sauvagarde.

Cette commande recharge un programme qui a été sauvegardé sur cassette. Il faudra mettre en route le magnétophone avant d'appuyer sur la touche NEWLINE. Le programme sera chargé quand le code 0/0 s'affichera.

Le programme sera rechargé tel qu'il a été sauvegardé; ainsi les variables reprendront les valeurs qu'elles avaient au moment de la sauvegarde.

Si le nom est omis, le programme chargé est le premier trouvé sur la cassette.

**Remarques**: 1. Si vous désirez relancer l'exécution de votre programme avec les valeurs rechargées des variables, il faudra utiliser la commande GOTO au lieu de RUN.

*Voir aussi commande RUN et instruction GOTO.*

2. L'interface cassette est relativement fiable, toutefois il faut trouver le bon niveau de sortie du magnétophone pour chaque cassette préenregistrée. Pour les utilisateurs du ZX 80, le niveau est différent pour la ROM 4K et 8K.



**NEW**: Commande.

NEW

Cette commande efface la mémoire et réinitialise les variables du système (qui assurent le fonctionnement interne du ZX 81).

Elle doit être utilisée:

— Avant l'entrée d'un nouveau programme BASIC.

— *Pour libérer la place en mémoire après modification de la variable RAMTOP (Voir chapitre ACCÈS AU LANGAGE MACHINE).*

**RUN**: Commande.

RUN n

Cette commande lance l'exécution de votre programme à partir de l'instruction dont le numéro de ligne est n, ou à la première instruction suivante si la ligne n n'existe pas.

Si n est omis ou a la valeur zéro l'exécution commence au début du programme.

Cette commande remet les variables de votre programme à zéro avant le début de l'exécution. Dans certains cas, on désire garder le contenu des variables, il faudra alors lancer l'exécution par la commande GOTO n.

**SAVE**: Commande.

SAVE "nom"

le nom ne peut pas comporter plus de 124 caractères.

Cette instruction permet de sauvegarder un programme sur cassette en lui donnant un nom. Il faudra, avant d'appuyer sur la touche NEWLINE, mettre le magnétophone en enregistrement. Le programme sera totalement sauvegardé quand le code 0/0 apparaîtra sur l'écran.

Le programme est sauvegardé avec les valeurs de toutes ses variables.

L'absence de nom provoque une erreur.

**SLOW**: Commande.

Dans ce mode de fonctionnement l'affichage est permanent contrairement au mode FAST, mais en contrepartie l'exécution du programme est quatre fois moins rapide.

Le ZX 81 affiche l'image sur l'écran en balayant ligne après ligne. Dans le mode SLOW, l'affichage sur l'écran est prioritaire. La gestion du clavier et l'exécution de votre programme ne se font que pendant le temps de retour du spot lumineux de la fin d'une ligne au début de la ligne suivante ! Le ZX 80 ne permet pas ce mode de fonctionnement (même avec la ROM 8K).

*Voir aussi instruction FAST et chapitre POSSIBILITÉ GRAPHIQUE, Paragraphe: GRAPHIQUES ANIMÉS.*

## 1.2. LES INSTRUCTIONS ET FONCTIONS DU ZX 81

**ABS**: Fonction.

ABS n
n est un nombre décimal.

La fonction ABS donne la valeur absolue du nombre n (ce qui revient à fixer un signe plus pour la valeur numérique de cette variable).

*Exemple:*

On désire s'assurer qu'un nombre est compris entre − 2 et + 2.

```
10  PRINT "ENTREZ UN NOMBRE COMPRIS ENTRE 2 ET — 2"
20  INPUT A
30  IF ABS A > 2 THEN GOTO 10
40  PRINT "OK"
```



**ACS, ASN, ATN**: Fonctions.

ACS n
ASN n  n est un nombre décimal.
ATN n

Les fonctions ACS, ASN, ATN permettent de calculer respectivement l'Arccosinus, l'Arcsinus, et l'Arctangente du nombre n. Le résultat sera exprimé en radians.

(Si Y = SIN X alors X = ASN Y avec X compris entre $-\pi/2$ et $+\pi/2$.)

**Remarque**: Pour les fonctions ASN et ACS le nombre n devra être compris entre − 1 et + 1 afin d'éviter une erreur de type A.

*Exemple:*

```
10   PRINT    "CALCUL D UN ARCSINUS"
20   PRINT    "ENTREZ VOTRE NOMBRE"
30   INPUT    A
40   IF ABS   A > 1 THEN GOTO 80
50   LET      B = ASN A
60   PRINT    "L ARCSINUS DE "; A;" EST UN ANGLE
              DE " ; B; " RADIANS."
70   STOP
80   PRINT    "VOTRE NOMBRE DOIT ETRE COMPRIS
              ENTRE − 1 ET + 1".
90   GOTO     20
```

**CHR$**: Fonction.

**CHR$** n
n est un nombre décimal.

La fonction CHR$ permet d'obtenir le caractère dont le code numérique est n. (Voir table des codes de l'annexe A du manuel SINCLAIR.)

*Exemple 1:*

```
10 PRINT CHR$ 38
```

provoquera l'affichage de la lettre A sur l'écran.

**Remarque:** Si n n'est pas un nombre entier, la valeur considérée sera celle du *nombre entier le plus proche.*

Ainsi:

```
10 PRINT CHR$ 38.3        affichera la lettre A
10 PRINT CHR$ 38.5        affichera la lettre B
10 PRINT CHR$ 38.7        affichera la lettre B
```

*Exemple 2:*

Le sous-programme suivant permet de savoir si l'utilisateur a répondu OUI ou NON à une question.

```
10 LET A = 0
20 PRINT "REPONDEZ OUI OU NON"
30 INPUT R$
40 IF CHR$ CODE R$ = " O " THEN LET A = 1
50 RETURN
```

Si l'utilisateur répond OUI la variable A prendra la valeur 1 et s'il répond NON elle prendra la valeur 0 (zéro).

*Exemple 3:*

Lorsque vous écrivez un programme en langage machine, vous pouvez vérifier que celui-ci commence bien à l'adresse voulue par la méthode suivante:

10 PRINT CHR$ (PEEK *adresse*)

*Adresse* est l'adresse décimale du début du sous-programme écrit en langage machine. L'instruction PRINT affichera alors le caractère, l'instruction ou la fonction correspondant au code hexadécimal du premier octet du sous-programme.

Ainsi pour l'exemple étudié dans le chapitre ACCÈS AU LANGAGE MACHINE, l'instruction affichée sera GOSUB dont le code numérique est 237 soit ED en hexadécimal.



**CODE**: Fonction.

CODE x
x est une variable alphanumérique.

La fonction CODE donne le code numérique du premier caractère contenu dans la variable alphanumérique x. Si cette variable alphanumérique est vide (elle a été initialisée à " ") la valeur retournée sera nulle.

*Exemple 1:*

```
10 LET A = CODE "ABCD"
20 PRINT A
```

affichera 38 qui est le code numérique de la lettre A.

*Exemple 2:* Reprenons l'exemple 2 de la fonction CHR$:

— Savoir si l'utilisateur a répondu OUI ou NON? Le code de la lettre O, première lettre du mot OUI, est 52.

Le sous-programme devient donc:

```
10 LET A = 0
20 PRINT "REPONDEZ OUI OU NON"
30 INPUT R$
40 IF CODE R$ = 52 THEN LET A = 1
50 RETURN
```

**COS, SIN, TAN**: Fonctions.

COS n
SIN n  n est un angle exprimé en radians
TAN n

Les fonctions COS, SIN, et TAN permettent de calculer respectivement le cosinus, le sinus et la tangente d'un angle dont la valeur est donnée en radians.

*Exemple :*

```
10 PRINT "ENTREZ D POUR DEGRES OU R POUR RADIANS
20 INPUT A$
30 PRINT "VALEUR DE L ANGLE?"
40 INPUT V
50 IF A$ = "D" THEN LET V = V * PI/180
60 PRINT "LE SINUS DE L ANGLE EST: " ; SIN V
70 STOP
```

**DIM** : Instruction.

DIM n ($m_1$, $m_2$, ...) — n : nom du tableau
— $m_1$, $m_2$, ... : nombre d'éléments de chaque dimension du tableau.

L'instruction DIM réserve la place en mémoire pour tous les éléments du tableau n. Ceux-ci seront initialisés à zéro.

— Le premier élément aura l'indice 1.

— Le nombre de dimensions n'est pas limité.

— Le nom du tableau ne devra comporter qu'une lettre. Il est toutefois *possible d'avoir un tableau et une variable avec des noms identiques.*

*Exemple 1 :*

```
10 DIM A(10)
20 FOR A = 1 TO 10
30 LET A(A) = A
40 NEXT A
50 STOP
```

Pour déclarer un tableau de variables alphanumériques, il faudra préciser en plus le nombre maximum de caractères d'une variable. De plus le nom du tableau devra être suivi du caractère **$**. *Un tableau et une variable alphanumériques ne pourront pas avoir le même nom.*

*Exemple 2 :*

```
10 DIM B$ (5,3)
20 LET B$ (1) = "AAA"
30 LET B$ (2) = "B"
```



Réservation d'un tableau alphanumérique de cinq éléments, chaque élément pouvant avoir trois caractères.

**Remarque :** Si l'indice n'est pas un nombre entier, il sera arrondi *au nombre entier le plus proche.* Ainsi :

```
A (1.4) = A(1)
A (1.5) = A(2)
A (1.7) = A(2)
```

**EXP** : Fonction.

EXP x
x est une variable numérique.

La fonction EXP calcule l'exponentielle de la variable numérique x.

*Exemple : Voir fonction LN.*

**FAST** : Instruction.

FAST

*Voir commande FAST.*

**FOR** : Instruction.

FOR I = VI TO VF STEP PP

I : variable de contrôle de la boucle
VI : valeur initiale de I
VF : valeur finale de I
PP : pas de progression (il peut-être négatif)

L'instruction FOR permet l'initialisation de la variable I de contrôle de la boucle ainsi définie. Cette variable ne doit comporter qu'une seule lettre ce qui limite le nombre de boucles imbriquées à 26 (nombre de lettres de l'alphabet).

*Si la valeur initiale est supérieure à la valeur finale, la boucle ne sera pas exécutée* contrairement aux autres BASIC avec lesquels le corps de la boucle (i.c. les instructions situées entre FOR et NEXT) sera exécuté au moins une fois.

VI, VF et PP peuvent être des expressions arithmétiques.

*Exemple 1 :*

```
10 FOR I = A + 2 TO (B * 5/2) STEP - 1
```

*Exemple 2 :*

```
10 FOR I = 0 TO 255
20 PRINT CHR$ I
30 NEXT I
```

*Exemple 3 :*

```
10 DIM A(50)
20 FOR B = 1 TO 50
30 LET A(B) = 10
40 NEXT B
```

*Voir aussi instruction NEXT.*

**GOSUB** : Instruction.

GOSUB n
n variable numérique.

L'instruction GOSUB permet de se brancher à un sous-programme dont la première instruction a le numéro n. Le numéro de l'instruction de retour est stockée en mémoire (dans une pile réservée à cet effet).

Comme pour l'instruction GOTO, n peut être un nombre entier, une variable ou une expression arithmétique.

*Exemple 1 :*

```
100 GOSUB 10
```



*Exemple 2 :*

```
150 GOSUB (V + 2) * B
```

*Exemple 3 :*

```
 10 REM VERIFICATION PARAMETRE
 20 INPUT A
 30 IF A > 0 AND A < 4 THEN GOSUB 200 + A * 10
 40 PRINT "ENTREZ UN NOMBRE COMPRIS ENTRE O ET 3"
 50 GOTO 20
    .
    :
200 REM A = 0

209 RETURN
210 REM A = 1

219 RETURN
220 REM A = 2

229 RETURN
230 REM A = 3

239 RETURN
```

Un sous-programme peut en appeler un autre. Le nombre de niveaux du sous-programme dépend uniquement de l'espace mémoire disponible. En effet chaque appel de sous-programme provoque la mémorisation de l'adresse de retour dans la pile "sous-programme". Cette pile augmente par adresse décroissante. Ainsi si sa taille devient trop importante il se peut qu'elle vienne écraser une partie de votre programme.

*La récursivité est autorisée* (un programme peut s'appeler lui-même). Ceci est tout à fait inhabituel dans le langage BASIC.

*Exemple 4 :* Calcul de factorielle n : (n!).

n! = n × (n − 1) × (n − 2) × ... × 2 × 1
4! = 4 × 3 × 2 × 1 = 24

```
 10 REM CALCUL DE FACTORIELLE N
 20 PRINT "EXEMPLE DE SOUS-PROGRAMME RECURSIF"
 30 LET A = 1
 40 PRINT "ENTREZ LA VALEUR DE N"
 50 INPUT N
 60 LET B = N
 70 GOSUB 100
 80 PRINT "LA VALEUR DE FACTORIELLE" ; B " EST" ; A
 90 STOP
 95 REM SOUS-PROGRAMME RECURSIF
100 LET A = A * N
110 LET N = N − 1
120 IF N > 1 THEN GOSUB 100
130 RETURN
```

**GOTO** : Instruction.

GOTO n

Saut inconditionnel à l'instruction ayant le numéro de ligne n, ou si n n'existe pas à la première instruction dont le numéro est supérieur à n.

n peut être { un nombre / une variable / une expression arithmétique.

Ces deux derniers cas sont propres au BASIC du ZX 81.

Ceci permet de pallier au manque de l'instruction : ON variable GOTO instruction 1, instruction 2, ... .

*Exemple 1 :*

```
30 GOTO 100
```



*Exemple 2 :*

```
10 LET A = 0
20 PRINT "ALLER EN ?"
30 INPUT A
40 GOTO A
```

*Exemple 3 :*

```
  10 LET R = 0
  20 PRINT " REPONDEZ OUI OU NON "
  30 INPUT T$
  40 IF T$ = " OUI " THEN LET R = 1
  50 LET R = R * 20
  60 GOTO (R + 100) * 10
   .
   .
1 000 REM VOUS AVEZ REPONDU NON
   .
   .
   .
1 200 REM VOUS AVEZ REPONDU OUI
   .
   .
9 000 STOP.
```

**IF — THEN** : Instruction.

IF Condition THEN Instruction.

Si la condition est vraie alors l'instruction suivant THEN est exécutée ; si elle est fausse, le contrôle est donné à l'instruction suivante.

*Exemple 1 :*

```
10 IF A = B THEN GOTO 100
```

*Exemple 2 :*

```
10 IF A$ = B$ AND A > B OR A$ = C$ AND A < C THEN GOTO
   1 000
```

*Exemple 3:*

```
900 IF S = 1 THEN GOTO 950
910 REM LA REPONSE EST NON
     .
     .
950 REM LA REPONSE EST OUI
```

**Remarque:** Lorsque votre ZX 81 évalue une condition logique il affecte à une variable interne la valeur 1 si la condition logique est vraie et la valeur zéro si elle est fausse. Ceci peut-être utilisé de la manière suivante:

Vous désirez tester si la valeur d'une variable est différente de zéro. Une solution possible est:

```
100 IF A <> O THEN GOTO 200
```

Cependant on peut aussi écrire:

```
100 IF A THEN GOTO 200
```

En effet votre ZX 81 comprendra ceci de la manière suivante:

— Si A est nul: la condition est fausse.

— Si A a une valeur *quelconque différente* de zéro ceci est interprété comme une condition vraie (même si A n'est pas égal à 1).

**INKEY$**: Fonction.

INKEY$ sans argument

La fonction INKEY$ permet de reconnaître la touche qui est pressée sur le clavier au moment où la fonction elle-même s'exécute.

*Exemple 1:*

Dans de nombreux jeux on utilise les touches →, ←, ↑ et ↓ pour provoquer le déplacement d'un objet (vaisseau spatial par exemple). Le petit programme suivant teste si la touche pressée correspond à l'une des flèches — (à savoir les touches 5, 6, 7, et 8 du clavier sans appuyer sur SHIFT), et dans ce cas déplace le curseur dans la direction voulue. L'exécution du programme pourra être arrêtée en appuyant sur la touche S.

```
 10 REM    TEST DES APPUIS TOUCHES
 20 LET    X = INT (RND * 64)
 30 LET    Y = INT (RND * 44)
 40 LET    X1 = X
 50 LET    Y1 = Y
 60 PLOT   X, Y
 70 IF     INKEY$ = "S" THEN STOP
 80 LET    TOUCHE = CODE INKEY$ — 32
 90 LET    X = ((TOUCHE = 1) * — 1) + (TOUCHE = 4) + X
100 LET    Y = ((TOUCHE = 2) * — 1) + (TOUCHE = 3) + Y
110 LET    IF X > 63 OR Y > 43 THEN GOTO 140
120 UNPLOT X1, Y1
130 GOTO   40
140 LET    X = X1
150 LET    Y = Y1
160 GOTO   70
170 STOP
```

*Exemple 2:*

Le programme suivant permet de tracer un dessin. Pour cela il faut utiliser les touches 5, 6, 7 et 8. A l'origine le curseur se situera en bas à gauche de l'écran. En appuyant sur la touche V qui sert aussi à la fonction CLS vous effacez l'écran.

```
 10 REM    DESSIN
 20 LET    X = 0
 30 LET    Y = X
 40 PLOT   X, Y
 50 LET    T = CODE INKEY$ — 32
 60 IF     T = 27 THEN CLS
 70 LET    X1 = ((T = 1) * — 1) + (T = 4)
 80 LET    Y1 = ((T = 2) * — 1) + (T = 3)
 90 IF     (Y1 + Y) > 43 OR (X1 + X) > 63 THEN GOTO 50
100 LET    X = X + X1
110 LET    Y = Y + Y1
120 GOTO   40
```

**INPUT** : Instruction.

INPUT x
x est un nom de variable.

L'instruction INPUT permet l'entrée de données lors de l'exécution de votre programme. Si votre ZX 81 est en mode FAST, l'affichage réapparaîtra. En effet lors de l'exécution d'un programme l'affichage disparaît car la priorité est donnée au calcul. En attendant l'introduction de votre donnée au clavier, votre ZX 81 n'effectue aucun calcul et il a donc le temps de "s'occuper" de l'affichage.

Il est possible de rentrer des données numériques, alphanumériques (dans ce cas le curseur apparaîtra entre guillemets pour vous le rappeler), des expressions arithmétiques ou le nom de variables déjà définies dans le programme (dans ce dernier cas et si la variable attendue est alphanumérique il faudra supprimer les deux guillemets qui encadrent le curseur).

**Remarque:** Si vous entrez STOP en réponse à une instruction INPUT l'exécution du programme sera arrêtée.

*Exemple:*

```
  10 LET A$=" X"
  20 PRINT "COMMENT VOUS APPELEZ
 VOUS?"
  30 PRINT "SI VOUS NE VOULEZ PA
S REPONDRE "
  40 PRINT "TAPEZ A$ EN ENLEVANT
 D""ABORD LES GUILLEMETS"
  50 INPUT B$
  60 PRINT "JE VOUS APPELLERAI D
ONC MR ";B$
  70 STOP
```



**INT** : Fonction.

INT n
n est nombre décimal.

La fonction INT permet d'obtenir la partie entière du nombre n. Le résultat sera toujours la valeur entière *immédiatement inférieure*. Ainsi:

```
     INT   5.6 =  5
et   INT - 5.6 = - 6
```

*Exemple:*

Il est souvent utile de pouvoir décomposer un nombre en deux autres nombres dont l'un est un multiple de 256. (Par exemple 16 509 serait décomposé en 64 × 256 + 125.) Ceci peut-être fait de la manière suivante:

```
10  PRINT   "ENTREZ UN NOMBRE POSITIF"
20  INPUT   N
30  IF      N < 0 THEN GOTO 10
40  LET     A = INT (N/256)
50  LET     B = N - A * 256
60  PRINT   N ; " SE DECOMPOSE EN:" ; A; " * 256 + " ; B
70  STOP
```

**LEN** : Fonction.

LEN x
x est une variable alphanumérique.

La fonction LEN donne la longueur de la variable alphanumérique x (i.e. le nombre de caractères qui la composent). Si cette variable est vide (i.e. initialisée à "") sa longueur sera zéro. Cette fonction est surtout utile pour éviter les erreurs lorsque l'on travaille avec ce type de variables.

*Exemple 1:*

Ce petit programme permet d'afficher un mot en diagonale.

```
10 INPUT A$
20 FOR I = 1 TO LEN A$
30 PRINT TAB I, A$ (I)
40 NEXT I
50 STOP
```

*Exemple 2:*

```
9 900  PRINT " NOM DU PROGRAMME? "
9 910  INPUT N$
9 920  IF LEN N$ = 0 THEN GOTO 9 900
9 930  SAVE N$
9 940  STOP
```

Le test de l'instruction 9 920 permet d'éviter le code erreur F qui apparaît lorsque vous essayez de sauvegarder un programme sans lui donner de nom.

**LET**: Instruction.

LET X = Y

— X est un nom variable numérique (ou alphanumérique).

— Y est une expression arithmétique ou un nom de variable numérique (ou alphanumérique).

L'affectation se fera toujours avec l'instruction LET. Certains BASIC permettent l'omission du mot-clé LET mais ce n'est pas le cas pour le ZX 81.

Lorsque l'affectation met en cause des variables alphanumériques de longueurs différentes, la règle suivante s'applique:

— si la variable réceptrice est plus petite que la variable émettrice, les derniers caractères de droite sont perdus,

— si la variable réceptrice est plus longue que la variable émettrice, la place restant à droite sera remplie avec des blancs (espaces).

*Exemple:*

```
10  DIM A$(3,3)
20  LET A$(1) = " ABCDEF "
30  LET A$(2) = " X "
40  LET A$(3) = " 123 "
50  PRINT A$(1); " ; " ; A$(2); " ; " ; A$(3)
```

Le résultat sera ABC; X ; 123



**LET** X = Y = Z

X : est une variable numérique
Y et Z : sont des variables numériques (ou alphanumériques).

Le BASIC de votre ZX 81 interprétera cette instruction de la manière suivante:

Affecter à X le résultat du test «Y est-il égal à Z ?».
Ainsi si Y = Z alors X prendra la valeur 1
si Y ≠ Z alors X prendra la valeur 0

Vous pouvez aisément vérifier cela avec les quelques instructions suivantes:

```
10  INPUT A
20  INPUT B
30  LET C = A = B
40  PRINT "C ="; C
50  STOP
```

**Remarque:**

Ceci peut aussi s'appliquer à toutes les fonctions logiques.

*Exemple:*

10 LET A = (4 AND (B = 3)) — donnera la valeur 4 à A si B = 3
— donnera la valeur 0 (zéro) à A si B ≠ 3

en effet supposons que B = 3.

L'instruction sera donc équivalente à:

10 LET A = (4 AND (3 = 3)) qui est la même chose que
10 LET A = (4 AND 1) qui est identique à
10 LET A = 4

**Remarque:** L'exemple 2 de la fonction CODE peut être programmé de la façon suivante:

```
20  PRINT   " REPONDEZ OUI OU NON "
30  INPUT   R$
40  LET     A = CODE R$ = 52
50  RETURN
```

**LN** : Fonction.

LN x
    x variable numérique dont la valeur est strictement positive.

La fonction LN calcule le logarithme népérien de x.

*Exemple :*

```
 10 PRINT   "CALCUL DE LA RACINE NIEME "
 20 PRINT   "D UN NOMBRE POSITIF"
 30 PRINT   "RANG DE LA RACINE?"
 40 INPUT   R
 50 PRINT   "NOMBRE?"
 60 INPUT   N
 70 LET     C = (LN N)/R
 80 LET     C = EXP C
 90 PRINT   "LA RACINE "; R; " EME DE "; N; " EST "; C
100 STOP
```

Voir aussi fonction EXP

**LOAD** : Instruction.

*Voir Commande LOAD.*

**NEXT** : Instruction.

NEXT I

L'instruction NEXT incrémente la variable de boucle de la valeur du pas de progression et donne le contrôle :

— à l'instruction suivante si la valeur de la variable de boucle est supérieure à la valeur finale,

— à la première instruction après l'instruction FOR sinon.

*Exemples :*

Voir instruction FOR.



**PAUSE** : Instruction.

PAUSE n

L'instruction PAUSE arrête l'exécution du programme pendant n/50ème de seconde. Si n est supérieur à 32 767 l'exécution du programme ne recommencera que si une touche est pressée.

Si vous exécutez votre programme en mode FAST n'oubliez pas d'exécuter l'instruction POKE 16 437,255 immédiatement après PAUSE (voir manuel SINCLAIR).

**PEEK** : Fonction.

PEEK n
    n est une adresse mémoire *en décimal.*

Cette fonction permet de lire le contenu de l'adresse mémoire n. Cette instruction est surtout utilisée pour aller lire le contenu de certaines variables du système et pour l'accès au langage machine.

*Exemple :*

Recherchons le numéro de ligne à partir duquel l'exécution est repartie. Le numéro se trouve dans la variable système OLDPPC située aux adresses 16 427 et 16 428.

```
05  STOP    ... (envoyer la commande CONT pour poursuivre l'exé-
            cution)
10  LET     A = PEEK (16 427)
20  LET     A = PEEK (16 428) * 256 + A
30  PRINT   "LE NUMERO DE L'INSTRUCTION EST:"; A
40  STOP
```

**Remarque :**

Vous pouvez lire le contenu de n'importe quelle adresse mémoire (ROM et RAM).

La ROM commence à l'adresse 0 et se termine à l'adresse décimale 8 191.

La RAM commence à l'adresse décimale 16 384 et se termine à l'adresse décimale :

— 17 407 pour la configuration 1K.

— 32 767 avec une extension 16K.

**PLOT** : Instruction.

PLOT x, y
x : numéro de colonne compris entre 0 et 63
y : numéro de ligne compris entre 0 et 43

L'instruction PLOT permet d'afficher sur l'écran un point aux coordonnées x et y. Pour plus de précisions voir le chapitre *POSSIBILITÉS GRAPHIQUES.*

*Voir aussi instruction UNPLOT.*

**POKE :** Instruction.

POKE n, m
n est une adresse *en décimal*
m est une quantité numérique comprise entre 0 et 255.

L'instruction POKE permet de ranger la valeur m dans la cellule mémoire ayant l'adresse n. Cette instruction est surtout utilisée pour la programmation en langage machine.

*Exemple :*

La séquence suivante fera boucler le programme sur l'instruction ayant le numéro de ligne 10 en venant écrire dans la variable système NXTLIN.

```
10 PRINT "ICI 10"
20 POKE  16 425, 125
```

*Voir aussi ACCÈS AU LANGAGE MACHINE et instruction PEEK.*



**Remarque :** L'adresse n devra référencer une adresse de la RAM (sinon, l'instruction POKE n'aura pas grand intérêt). Ainsi n devra être compris entre

16 384 et 17 407 pour la configuration 1K et entre

16 384 et 32 767 pour la configuration 16K

**PRINT** : Instruction.

PRINT $a_1 \left\{ \begin{matrix} ; \\ , \end{matrix} \right\}$ $a_2 \left\{ \begin{matrix} ; \\ , \end{matrix} \right\}$ $a_3$ ...

L'instruction PRINT permet l'affichage de résultats sur l'écran. La virgule provoquera une tabulation automatique de l'affichage toutes les 16 colonnes, alors que le point-virgule donne un affichage contigü. Les arguments peuvent être des expressions arithmétiques ou des variables alphanumériques.

*Exemple :*

```
PRINT "LA VALEUR TROUVEE EST:"; A + 2
```

PRINT AT m, n ;

Le résultat est affiché sur la ligne m à partir de la colonne n.
m doit être compris entre 0 et 21, la ligne 0 étant la première ligne en haut de l'écran.
n doit être compris entre 0 et 31, la colonne 0 étant la colonne la plus à gauche de l'écran.

PRINT TAB n

n numéro de colonne (modulo 32).

Ceci provoque l'affichage du résultat sur la ligne courante à partir de la colonne n.

La clause TAB ne permet pas de revenir en arrière sur la même ligne.

Ainsi :

```
100 PRINT TAB (27); "*"; TAB (10); "A"
```

provoquera l'affichage de deux lignes, la première ayant l'astérisque en colonne 27 et la seconde le caractère A en colonne 10.

Pour ces deux clauses (AT et TAB) la dernière ligne de l'écran est la ligne 21, les lignes 22 et 23 étant réservées pour les commandes.

On se reportera également au chapitre GRAPHIQUE pour l'utilisation de l'instruction PRINT.

**RAND** : Fonction.

RAND n
 n nombre entier compris entre 1 et 65 535

La fonction RAND permet d'initialiser le processus de génération des nombres aléatoires.

RAND 0 ou RAND sans argument ont pour effet d'initialiser le processus avec le contenu de la variable système FRAMES (voir description des variables système dans le manuel SINCLAIR p. 179).

**REM** : Instruction.

REM Chaîne de caractères.

L'instruction REM permet d'introduire des commentaires dans un programme. Les caractères contenus dans cette instruction ne seront pas analysés.

Utilisée de manière astucieuse cette instruction permet de réserver de la place en mémoire pour venir écrire un programme en langage machine (voir chapitre *ACCÈS AU LANGAGE MACHINE*).



**RETURN** : Instruction.

RETURN

L'instruction RETURN doit obligatoirement être la dernière instruction d'un sous-programme. Elle provoque le retour au programme principal.

*Exemple : Voir instruction GOSUB.*

**RND** : Fonction

RND

La fonction RND n'a pas d'arguments, elle permet d'engendrer un nombre "au hasard" compris entre zéro et un (0 et 1), la valeur 1 ne pouvant jamais être atteinte.

**SAVE** : Intruction.

Voir commande SAVE.

*Exemple :*

```
 10  REM    PROGRAMME XXXX
      .
      .
      .
200  PRINT  "NOM DU PROGRAMME ?"
210  INPUT  N$
220  PRINT  " METTEZ VOTRE MAGNETOPHONE EN MARCHE"
230  PRINT  "PUIS APPUYEZ SUR UNE TOUCHE DU CLAVIER
240  PAUSE  32 800
250  SAVE   N$
      .
      .
      .
```

**SGN** : Fonction.

SGN n
n est une variable numérique.

La fonction SGN (fonction signe) teste la valeur de la variable n et donne comme résultat.

– 1 si sa valeur est négative.
0 si sa valeur est nulle.
et + 1 si sa valeur est positive.

*Exemple 1 :*

```
10   REM     ESSAI DE LA FONCTION SGN
20   DIM     A(3)
30   LET     A(1) = 0
40   LET     A(2) = – 5
50   LET     A(3) = 5
60   FOR     I = 1 TO 3
70   PRINT   " POUR A = "; A(I); " LE RESULTAT DE LA FONC-
             TION SGN EST: "; SGN A(I)
80   NEXT I
90   STOP
```

*Exemple 2 :*

Le sous-programme suivant arrondit un nombre à deux chiffres après la virgule.

Le nombre à arrondir devra se trouver dans la variable A.

```
9 000  REM     ARRONDI
9 010  LET     A3 = 100
9 020  LET     A1 = INT(ABS A + 0.005) * SGN A
9 030  LET     A2 = INT (ABS (A – A1) * A3 + 0.5) * SGN A
9 040  LET     A  = A1 + A2/100
9 050  RETURN
```



**SLOW** : Instruction.

*Voir commande SLOW.*

**STOP** : Instruction.

STOP

Cette instruction provoque l'arrêt du programme. Celui-ci pourra être relancé par la commande CONT.

Si cette commande est frappée en réponse à une instruction INPUT il y aura arrêt du programme avec le code d'erreur D.

**UNPLOT** : Instruction.

UNPLOT x, y
x : numéro de colonne compris entre 0 et 63
y : numéro de ligne compris entre 0 et 43

L'instruction UNPLOT permet "d'éteindre" le point de coordonnées x, y.

*Voir aussi instruction PLOT et chapitre POSSIBILITÉS GRAPHIQUES.*

**STR$:** Fonction.

STR$ x
x est une variable numérique.

La fonction STR$ permet de transformer une variable numérique en variable alphanumérique.

*Exemple:*

```
10  INPUT   A
20  LET     B$ = STR$ A
30  PRINT   B$
40  STOP
```

éditera la valeur du nombre que vous avez entré.

**VAL:** Fonction.

VAL variable alphanumérique.

VAL permet l'évaluation d'une expression algébrique définie comme une chaîne de caractères.

La fonction VAL permet d'effectuer des calculs à l'intérieur d'une variable alphanumérique. Le résultat est obtenu en évaluant la variable alphanumérique comme une expression arithmétique.

*Exemples:*

**1.**

```
10  LET     A$ = " ASN (SQR 2/2) * 4 "
20  PRINT   " A = "; VAL A$
```

la valeur de A sera 3,1415... (π).



**2.** Grâce au programme suivant vous pourrez calculer la valeur de n'importe quelle expression.

(Pour arrêter le programme il faut appuyer sur la touche NEWLINE en réponse à la question expression à évaluer?)

```
10  REM     EVALUATIONS D'EXPRESSIONS
20  PRINT   " EXPRESSION A EVALUER ? "
30  INPUT   E$
40  IF      E$ = " " THEN GOTO 80
50  LET     R = VAL E$
60  PRINT   " LE RESULTAT EST: "; R
70  GOTO    20
80  STOP
```

**Remarques:**

**1.** Avec les instructions PRINT AT, PLOT et UNPLOT la fonction VAL ne peut apparaître que dans la première des coordonnées.

Ainsi l'instruction:

```
100  PRINT   AT VAL "X", VAL "Z"
```

devra être changée en:

```
100  LET     Y = VAL "Z"
105  PRINT   AT VAL "X", Y
```

**2.** Si la fonction VAL fait partie d'une expression arithmétique, elle devra se trouver en tête de cette expression.

Ainsi l'instruction 800 LET X = A ** 2 + VAL "C"

doit être écrite: 800 LET X = VAL "C" + A ** 2

## Variables

Le ZX 81 accepte quatre types de variables :

— les variables numériques,
— les variables alphanumériques,
— les éléments de tableau,
— les variables de contrôle de boucle.

1. *Les variables numériques*

Les noms de ces variables ne sont pas limités à un nombre fixe de caractères. La seule restriction est que le premier caractère doit être alphabétique. Ces noms peuvent comporter des espaces (ils seront ignorés par le BASIC du ZX 81). Il faut cependant avoir à l'esprit que le ZX 81 stocke le nom complet de la variable en mémoire ; ainsi en donnant des noms de variables trop longs, vous risquez de diminuer fortement la place disponible en mémoire.

2. *Les variables alphanumériques*

Leur nom se limite à une seule lettre suivi du caractère $ (dollar).

3. *Les éléments de tableau*

Leur nom consiste en une seule lettre suivi du numéro de l'élément désiré.

Si le tableau est un tableau alphanumérique il faudra faire suivre le nom du caractère $.

*Exemples :*

```
A (7)
A$ (2)
```

4. *Les variables de contrôle de boucle*

Les noms de ces variables ne doivent comporter qu'une seule lettre.

**Remarque :** Il est possible de donner le même nom à une variable numérique et à un tableau, le ZX 81 distinguera parfaitement les deux.
Par contre ceci ne s'applique pas pour les variables et les tableaux alphanumériques.



## Priorité des opérateurs : Règle

Toute expression arithmétique ou logique est évaluée de gauche à droite selon les priorités suivantes des opérateurs.

| | | |
|---|---|---|
| Évaluation des fonctions | | 11 |
| Élévation à la puissance | ** | 10 |
| Opposé | – | 9 |
| Multiplication — division | *, / | 8 |
| Addition, soustraction | +, – | 6 |
| Comparaison | >, <, =, >=, =<, <> | 5 |
| Opération logique NON | NOT | 4 |
| Opération logique ET | AND | 3 |
| Opération logique OU | OR | 2 |

**Remarque :** L'évaluation peut être changée par l'emploi de parenthèses.

*Exemple :*

```
10 LET A = B * C – F + D/E
```

avec

```
             B = 3,  C = 4
   D = 6,    E = 2,  F = –3
```

sera évalué comme suit

```
10 LET A = 3 * 4   – (– 3) + 6/2
10 LET A =  12  – (– 3) + 3
10 LET A =  18
```

alors que :

```
10 LET A = B * (C – F) + D/E          donnera
10 LET A = 3 * (4 – (– 3)) + 6/2
         = 3 * 7 + 3
         = 24
```

# 2

# Quelques astuces

## 2.1. RÉDUIRE L'OCCUPATION MÉMOIRE

Dans sa version de base, le ZX 81 ne possède que 1 Koctets de mémoire vive. Enlevez-lui 125 octets pour les variables systèmes plus 25 octets pour le buffer d'affichage (dans sa version la plus réduite possible), vous réalisez alors rapidement que le problème de l'occupation mémoire est crucial pour ceux qui n'ont pas fait l'acquisition d'un bloc d'extension mémoire 16 K RAM! Dans ce paragraphe nous envisagerons diverses méthodes pour vous permettre de gagner quelques octets par ci, par là. Mais ne vous attendez pas à des miracles! Le gain de mémoire consiste souvent à "*grapiller*" un petit nombre d'octets dans telle ou telle partie de programme et à répéter cette opération plusieurs fois.

Avant tout, comment l'interpréteur BASIC stocke-t-il une instruction en mémoire? Le chapitre 27 du manuel SINCLAIR nous donne tous les renseignements nécessaires.

Chaque instruction occupe en mémoire:

— deux octets mémorisant le numéro de ligne de l'instruction,

— deux octets mémorisant sa longueur,

— un nombre d'octets égal au nombre de caractères qui composent l'instruction (vous remarquerez toutefois que les mots-clés comme INPUT, RETURN, etc... ne prennent qu'un octet),

— six octets mémorisant une constante numérique (si l'instruction en comporte une). Ces six octets se répartissent comme suit :

* un octet "initialisé" à 126 et servant de caractère drapeau,
* cinq octets mémorisant la valeur numérique en virgule flottante (1 octet pour l'exposant et 4 pour la mantisse),

— un octet pour le caractère retour à la ligne (NEWLINE = 118).

Prenons quelques exemples et introduisons les deux programmes suivants :

**1.**

```
  10   LET   A = 0
  20   LET   W = 1
  22   LET   B = A
  30   DIM   C (2,2)
3 000  LET   C (1,1) = 18
4 000  LET   F$ = " ZX 81 "
6 000  IF    A = 2 THEN LET B = 4
```

**2. Ce deuxième programme va afficher sur l'écran la manière dont sont stockée les instructions 10 à 6 000.**

```
8 990  LET    Z = 16 509
9 000  FOR    X = 0 TO 6
9 010  LET    T = PEEK (Z + 3) * 256 + PEEK (Z + 2) + 4
9 020  FOR    D = 0 TO T - 1
9 030  PRINT  PEEK (Z + D); " ";
9 040  NEXT   D
9 050  PRINT
9 060  PRINT
9 070  LET    Z = Z + T
9 080  NEXT   X
9 090  STOP
```

Nous aurons le résultat suivant :

```
  0  10 11   0 241 38 20 28 126 0 0 0  0  0  118
  0  20 11   0 241 60 20 29 126 129 0 0 0 0 118
  0  22  5   0 241 39 20 38 118
  0  30 20   0 233 40 16 30 126 130 0 0 0 0 26 30 126
130 0 0 0 0 17 118
 11 184 29   0 241 40 16 29 126 129 0 0 0 0 26 29 126
129 0 0 0 0 17  20  29  36  126 133 16 0 0 0 118
 15 160 14   0 241 43 13 20  11   0 63 61  0 36 29 0 11 118
 23 112 22   0 250 38 20 30 126 130  0  0  0 0 222 241
 39 20  32 126 131 0 0 0 0 118
```



### 2.1.1. Éviter les constantes numériques

Prenons l'instruction la plus longue de l'exemple :

```
3 000  LET C (1,1) = 18
```

Nous retrouvons :

— Le numéro de ligne dans les deux premiers octets stockés dans le sens *Poids forts* : 1[er] octet, *Poids faibles* : 2[e] octet. Ceci se vérifie aisément :

$$11 \times 256 + 184 = 3\,000$$

— La longueur de l'instruction sous la forme *Poids faibles* : 1[er] octet, *Poids forts* : 2[e] octet. Ainsi la longueur est de 29 octets : 29 + 0 × 256 = 29 (vous remarquerez que celle-ci ne comprend pas les octets qui mémorisent le numéro et la longueur de l'instruction). La longueur *exacte* sera donc obtenue en ajoutant 4 à cette dernière (voir programme de renumérotation et décodage d'un programme au chapitre 9).

— L'instruction elle-même : Vous constaterez que les constantes numériques sont mémorisées de deux manières différentes.

1. Chaque chiffre de la constante occupe un octet ainsi pour le nombre 18 nous avons un octet pour mémoriser le 1 (29) et un octet pour le 8 (36).
2. Sous la forme virgule flottante (caractère 126 suivi de 5 octets).

— Le caractère retour à la ligne (118).

Ainsi, sur 33 octets, 18 sont occupés pour mémoriser la valeur des constantes numériques (plus de la moitié ! De plus sur ces 18 octets, 6 sont réellement utilisés, les 12 autres sont à zéro !) Il faut donc chercher à *réduire* au maximum l'*utilisation des constantes numériques* dans un programme.

*Exemple:*

Considérons le programme de renumérotation au chapitre 9. Nous aurions pu supprimer la ligne 8 960 LET C = 256 et écrire les lignes 9 070, 9 080, 9 500 et 9 510 de la manière suivante:

```
9 070  POKE   W, INT (NL/256)
9 080  POKE   X + 1, NL - 256 * INT (NL/256)

9 500  LET    A = PEEK (W + 1) + 256 * PEEK W
9 510  LET    B = 4 + PEEK (W + 2) + 256 * PEEK (W + 3)
```

Le fonctionnement du programme n'en serait en rien modifié. Toutefois évaluons la place occupée en mémoire en remplaçant C par 256.

| | | | |
|---|---|---|---|
| ligne | 9 070 | → 2 + 2 + 11 + 6 + 1 + 1 | = 22 octets |
| ligne | 9 080 | → 2 + 11 + 6 + 9 + 6 + 1 + 1 | = 38 octets |
| ligne | 9 500 | → 2 + 2 + 8 + 6 + 5 + 6 + 3 + 1 | = 33 octets |
| | 9 510 | → 2 + 2 + 4 + 6 + 6 + 6 + 5 + 6 + 6 + 6 + 1 1 | = 51 octets |

Ces quatre instructions occupent donc 144 octets alors que dans le programme *avec la ligne supplémentaire* nous arrivons à un encombrement de:

| | | | | | |
|---|---|---|---|---|---|
| n° de la ligne | 8 960 | 9 070 | 9 080 | 9 500 | 9 510 |
| nbre d'octets | 17 | 15 | 28 | 25 | 43 |

soit 128 octets au total

Ainsi malgré la ligne supplémentaire nous sauvegardons 16 octets dans la zone programme. Nous rajoutons par contre une variable et donc 6 octets dans la zone de variables. Notre gain total sera alors 10 octets. Cependant chaque fois que nous réutiliserons la variable par la suite, nous économiserons 6 octets supplémentaires!



### 2.1.2. Utiliser les facilités de votre interpréteur BASIC!

Le BASIC de votre ZX 81 possède de nombreuses fonctions et quelques particularités que l'on peut exploiter pour gagner un peu de place en mémoire.

- CODE — Au lieu d'écrire 10 LET A = 15 qui occupe 16 octets, il vaut mieux écrire 10 LET A = CODE "?" qui occupe 12 octets.
- GOTO ET GOSUB — Le numéro de l'instruction à laquelle vous devez vous brancher peut-être contenu dans une variable. Si vous appelez souvent le même sous-programme il peut-être intéressant de faire GOSUB N en initialisant N auparavant.

- **Évaluation des expressions logiques** — Le BASIC du ZX 81 affecte à une variable interne la valeur 1 si une condition est vraie. Cette valeur interne peut être affectée à une variable du programme. Il suffit pour cela de faire LET A = (condition logique) par exemple 10 LET A = (B > 23). Ainsi si une variable B a déjà été définie dans le programme on pourra écrire:

| | | |
|---|---|---|
| 10 LET A = B = B | au lieu de | 10 LET A = 1 |
| 11 octets | | 15 octets |

De même on écrira:

| | | |
|---|---|---|
| 20 LET A = B − B | au lieu de | 20 LET A = 0 |
| 11 octets | | 15 octets |

De même, (p. 71 du manuel SINCLAIR), il est précisé que la condition peut être une expression arithmétique. Si la valeur de cette expression est nulle ceci sera considéré comme une condition fausse, toute autre valeur étant considérée comme une condition vraie. Ainsi au lieu d'écrire:

IF A = 0 THEN instruction, il faudra écrire IF NOT A THEN instruction

et

IF A <> 0 THEN instruction, il faudra écrire IF A THEN instruction

- **Nom des variables :** Le BASIC du ZX 81 ne limite pas la longueur du nom d'une variable. Toutefois il est préférable de ne donner, dans la mesure du possible, que des *noms ne comportant qu'une seule lettre* (A, B, C, ...). En effet le nom est stocké en entier dans la zone programme *et* dans la zone variable.

*Exemple :*

Vous désirez mémoriser un résultat dans une variable, envisageons les deux "méthodes" suivantes :

| ZONE PROGRAMME | Nbre octets | ZONE VARIABLES | Nbre octets |
|---|---|---|---|
| 100 LET R =A*B | 11 | R Valeur | 6 |
| 100 LET RESULTAT =A*B | 18 | RESULTAT Valeur | 13 |

la première méthode occupera 17 octets et la seconde 31.

- **Sous-programmes écrits en langage machine**

Cette méthode est sûrement celle qui vous apportera le plus d'économie mémoire surtout si la fonction que vous désirez réaliser n'est pas facilement programmable en BASIC. Pour plus de détails voir le chapitre "*ACCÈS AU LANGAGE MACHINE*".

**Remarque :** Nous avons vu précédemment qu'une constante numérique était stockée de deux manières. La première sert pour le listing et la seconde lors de l'exécution du programme (celle-ci sera plus rapide car la conversion en virgule flottante ayant déjà été faite, le ZX 81 n'aura pas à la refaire). Vous pouvez alors faire la petite chose amusante suivante.

— Rentrer le programme.

```
 10  PRINT   " ICI 10 "
 20  POKE    16 564,33
 30  STOP
 40  GOTO    140
140  PRINT   "ICI 140"
145  GOTO    160
150  PRINT   "ICI 150"
160  STOP
```



L'exécution s'arrête en 30, listez le programme. La ligne 40 est devenue 40 GOTO 150. Appuyer sur CONT. ICI 140 est affiché ! Ceci est dû au fait que la ligne 20 vient juste modifier le 4 en 5 du GOTO 140 mais elle ne touche pas à la valeur en virgule flottante.

## 2.2. CHAÎNAGE DE PROGRAMMES

### 2.2.1. Principe de fonctionnement

Il est possible de chaîner des programmes sauvegardés sur cassette magnétique et de passer le contrôle de l'un à l'autre. Pour cela il faudra *faire commencer votre programme par une instruction qui le sauvegarde lui-même.* Ainsi, quand vous rechargerez votre programme il reprendra l'exécution de lui-même. Si vous ne faites pas cela, le programme se chargera mais vous serez obligés de démarrer son exécution par la commande RUN (ou GOTO) ! Ceci s'explique de la façon suivante : Lorsque le programme est sauvegardé, sur la cassette se trouvera à la fois le programme et le contenu des variables, notamment des variables système. Le ZX 81 mémorise entre autre, dans ces variables système, l'adresse de l'instruction à exécuter après l'instruction SAVE. Ainsi lorsque vous rechargerez votre programme, l'adresse de la prochaine instruction à exécuter sera celle qui avait été sauvegardée, c'est-à-dire l'adresse de l'instruction suivant SAVE.

Prenons un exemple.

Considérons les deux programmes suivants :

*Programme 1*

```
 5  STOP
10  SAVE    PROG1
20  PRINT   "PROGRAMME NUMERO 1"
30  PRINT   "INSEREZ LA CASSETTE PROG2"
40  PRINT   "APPUYEZ SUR LA TOUCHE C LORSQUE VOUS SEREZ PRET"
50  IF      INKEY $ <> "C" GOTO 50
60  CLS
70  LOAD    "PROG2"
```

*Programme 2*

```
 5  STOP
10  SAVE    "PROG2"
20  PRINT   "PROGRAMME NUMERO 2"
30  PRINT   "INSEREZ LA CASSETTE PROG1"
40  PRINT   "APPUYEZ SUR LA TOUCHE C LORSQUE VOUS SEREZ PRET"
50  IF      INKEY $ <> "C" GOTO 50
60  CLS
70  LOAD    "PROG1"
```

Sauvegardez le premier programme sur cassette en lançant son exécution par RUN. Dès que la ligne "APPUYEZ... PRÊT" apparaît, appuyez sur la touche BREAK. L'adresse de la prochaine instruction à aller exécuter, mémorisée sur la cassette, est donc l'adresse de l'instruction :

```
20 PRINT ...
```

Lancez l'exécution du programme "PROG2" et procédez comme indiqué sur l'écran. Le programme "PROG1" sera donc chargé à la place de PROG2, et son exécution redémarrera directement à partir de l'instruction 20 (vous pouvez constater que l'instruction 5 STOP n'est pas exécutée).

### 2.2.2. Passage de paramètres

Quand un programme est rechargé, les variables du programme précédent sont écrasées. Il faut donc ranger les valeurs des variables que l'on désire passer au programme à charger dans une zone protégée, donc dans le haut de la mémoire par modification de la variable système RAMTOP. (Voir chapitre *ACCÈS AU LANGAGE MACHINE.*) Le plus simple est de mettre les valeurs que l'on désire sauvegarder en tête de la zone VARIABLES, puis de tout transférer dans la zone protégée. Le programme rechargé devra redéclarer les variables **dans le même ordre** que le programme qui a assuré la sauvegarde.



*Exemple :*

```
 10  SAVE   "PROG1"
 20  LET    A = 1
 30  LET    B = 2
 40  LET    C = 3
 50  LET    R = PEEK 16 388 + 256 * PEEK 16 389
 60  LET    V = PEEK 16 400 + 256 * PEEK 16 401
 70  FOR    I = 0 TO 17
 80  POKE   (R + I), PEEK (V + I)
 90  NEXT   I
100  LOAD   "PROG2"
```

Les instructions 50 à 90 permettent de venir ranger le contenu des 18 premiers octets de la zone VARIABLES dans la zone "protégée".

```
 10  SAVE   "PROG2"
 20  LET    A = 0
 30  LET    B = 0
 40  LET    C = 0
 50  LET    R = PEEK 16 388 + 256 * PEEK 16 389
 60  LET    V = PEEK 16 400 + 256 * PEEK 16 401
 70  FOR    I = 0 TO 17
 80  POKE   (V + I), PEEK (R + I)
 90  NEXT   I
100
            suite du programme
```

Les instructions 50 à 90 permettent de récupérer les 18 premiers octets de la zone "protégée" et de les ranger dans la zone VARIABLES.

# 3

# Accès au langage machine

La programmation en langage machine consiste à programmer directement le microprocesseur Z 80 qui constitue le cerveau de votre ZX 81. Pour la compréhension de ce chapitre, la programmation en assembleur Z 80 est supposée connue. Ceux qui ne la connaissent pas et désirent l'apprendre pourront se reporter à un autre ouvrage de la collection (à paraître).

## 3.1. PROGRAMMATION EN ASSEMBLEUR

Le ZX 81 ne permet pas, au contraire de bien d'autres micro-ordinateurs, la programmation en langage assembleur Z 80, ce qui vous oblige à introduire vous-même à l'endroit voulu le code des instructions à exécuter. Pour remédier à ce petit inconvénient, la société BUG-BYTE a développé un programme assembleur pour le ZX 81 (ZXAS: ZX ASsembler). Ce programme vendu sur cassette occupe 5 K en mémoire RAM (de 27 648 à 32 767, il vous faut donc obligatoirement une extension 16 K) et vous permet de rentrer directement les mnémoniques de l'assembleur du Z 80 en les incluant dans des instructions REM, dans vos programmes BASIC ; plusieurs instructions assembleur sont acceptées par instruction REM. Cet assembleur est un assembleur à deux passages. De plus, vous pouvez obtenir un listage du résultat de l'assemblage.

## 3.2. CRÉATION D'UN PROGRAMME EN LANGAGE MACHINE

Le ZX 81 n'a pas été conçu pour permettre la programmation en langage machine. Cependant, il existe trois méthodes permettant de créer un programme en langage machine à partir de l'utilisation astucieuse de l'interpréteur BASIC.

1. Emploi de l'instruction REM.
2. Utilisation d'un tableau alphanumérique.
3. Modification de la variable système RAMTOP.

Ces méthodes consistent à exploiter les réservations mémoire qu'effectuent certaines instructions BASIC, puis à utiliser les cases ainsi réservées à l'aide d'instructions PEEK et POKE ou de fonction de manipulation de caractères (CHR$).

### 3.2.1. Emploi de l'instruction REM

Avec cette méthode, votre programme BASIC devra *obligatoirement commencer par une instruction REM*, comportant un nombre de caractères au moins égal au nombre d'octets de votre programme en langage machine. Il vous suffit alors de venir écraser ces caractères un par un en utilisant l'instruction POKE.

L'adresse du premier caractère de l'instruction REM est 16 514 (16 509 adresse de la première instruction du programme plus 5: 2 octets pour le numéro de l'instruction, 2 octets pour sa longueur et 1 pour le code de l'instruction REM).

*Exemple:*

Le programme suivant donne l'abscisse x du dernier point affiché sur l'écran par l'instruction PLOT.

| *Mnémoniques* | *Décimal* | *Hexadécimal* | |
|---|---|---|---|
| LD BC, (COORDS) | 237, 75, 54, 64 | ED 4B 36 40 | : obtient les coordonnées x et y (x dans C, y dans B) |
| LD B,O | 06 0 | 06 00 | : force y à zéro |
| RET | 201 | C9 | : retour au programme BASIC |



**Remarques:**

**1.** Pour la conversion des mnémoniques en code décimal on peut se reporter à la table de l'annexe du manuel SINCLAIR.

**2.** COORDS = adresse 16 438 en décimal = adresse 4 036 en hexadécimal.

Le programme ci-dessus comporte 7 octets. Pour lui réserver une place en mémoire, on utilisera donc une instruction REM avec 7 caractères (chaque caractère réservant un octet). Ces caractères peuvent être quelconques car ils vont être écrasés par notre programme.

Le programme BASIC sera:

```
 10  REM    XXXXXXX
 20  LET    A= 16 514
 30  POKE   A,237
 40  POKE   A + 1,75
 50  POKE   A + 2,54
 60  POKE   A + 3,64
 70  POKE   A + 4,6
 80  POKE   A + 5,0
 90  POKE   A + 6,201
100  PLOT   25,30
110  LET    X= USR (A)
120  PRINT  " L ABSCISSE EST : "; X
130  STOP
```

Pour obtenir l'ordonnée y il faut rajouter au programme, avant la mise à zéro du registre B, l'instruction suivante:

| *Mnémonique* | *Décimal* | *Hexadécimal* | |
|---|---|---|---|
| LD C,B | 72 | 48 | : Recopie le contenu de B dans C. |

Ceci porte la longueur du programme à 8 octets. Nous ajoutons donc un caractère supplémentaire dans l'instruction REM. Le programme BASIC deviendra:

```
10   REM      XXXXXXXA
20
 .
 .
 .     inchangées
 .
 .
60
65   POKE     A + 4,72
70   POKE     A + 5,6
80   POKE     A + 6,0
90   POKE     A + 7,201
100  PLOT     25, 30
110  PRINT    "L ORDONNEE EST:"; USR (A)
120  STOP
```

Ainsi avec cette méthode :

— le sous-programme se trouve à une place fixe en mémoire. Son adresse ne dépend pas de la taille du programme BASIC,

— le sous-programme peut être sauvegardé avec le programme BASIC,

— dès que le programme a été créé, il est possible de détruire les lignes 30 à 90 (instruction POKE) permettant ainsi un gain de place en mémoire.

L'inconvénient de cette méthode est que l'affichage risque d'être quelque peu incohérent lorsque vous demandez un listage de votre programme. En fait le ZX 81 essaye d'afficher les caractères qui correspondent aux codes trouvés après l'instruction REM. Ainsi après avoir exécuté votre programme, vous verrez apparaître après l'instruction REM :

```
GOSUB?QRND ▚ TAN
```

Ceci disparaît si vous faites lister le programme à partir de l'instruction suivant l'instruction REM.



### 3.2.2. Utilisation d'un tableau alphanumérique

Ce tableau devra obligatoirement se trouver *au début de la zone mémoire réservée pour le stockage* des variables. Ceci s'obtient en plaçant une instruction DIM en tête du programme.

En réalité cette instruction DIM peut être précédée par d'autres instructions du moment que celles-ci ne comportent pas de déclaration de variables.

L'adresse de début du sous-programme est calculée ainsi :

```
LET A = PEEK 16 400 + PEKK 16 401 * 256 + 6
```

(6 octets étant nécessaires à la définition d'un tableau alphanumérique).

1 octet pour le nom du tableau
2 octets pour le nombre total d'éléments
1 octet pour le nombre de dimension
et 2 octets pour la dimension du tableau.

L'exemple précédent devient ainsi :

```
10   DIM     B$ (7)
20   LET     A = PEEK 16 400 + PEEK 16 401 * 256 + 6
30   LET     B$ (1) = CHR$ 237
40   LET     B$ (2) = CHR$  75
50   LET     B$ (3) = CHR$  54
60   LET     B$ (4) = CHR$  64
70   LET     B$ (5) = CHR$   6
80   LET     B$ (6) = CHR$   0
90   LET     B$ (7) = CHR$ 201
100  PLOT    25,30
110  PRINT   " L'ABSCISSE EST : "; USR (A)
120  STOP
```

Avec cette méthode :

— le sous-programme peut être sauvegardé en même temps que le programme BASIC, autorisant de ce fait la suppression des instructions de génération du programme en langage machine, sauf le calcul de l'adresse,

— la fonction CHR$ peut être remplacée par l'instruction POKE.

Par contre :

— le programme ne doit pas être lancé par RUN après une sauvegarde. En effet, ceci remettrait à zéro toutes les variables, y compris le tableau. Il faut donc utiliser la commande GOTO (GOTO 20 pour notre exemple). De même n'utilisez pas la commande CLEAR.

— La zone mémoire réservée au stockage des variables ne commence pas à une adresse fixe, contrairement à la zone mémoire programme. Pour les débranchements on ne pourra donc utiliser que les débranchements relatifs.

### 3.2.3. Modification de la variable système RAMTOP

Lors de la mise sous tension, le ZX 81 examine la capacité mémoire dont il dispose, et initialise la variable système RAMTOP avec l'adresse du dernier octet de la mémoire plus un.

Toutes les adresses supérieures à celles contenues dans RAMTOP, sont ignorées par le ZX 81 après une commande NEW.

Il suffit alors de modifier le contenu de la variable RAMTOP pour obtenir un espace mémoire dans lequel on pourra écrire un programme en langage machine puis d'exécuter une commande NEW. Ce programme ne pourra plus être détruit par une mauvaise commande comme cela était le cas avec les méthodes précédentes.

Ceci signifie aussi que vous ne pourrez plus sauvegarder votre programme en langage machine avec le reste du programme BASIC.

Reprenons l'exemple précédent. Pour réserver les 7 octets dont nous avons besoin, nous pouvons utiliser le programme suivant :



```
10  LET    C = PEEK 16 388 + 256 * PEEK 16 389
20  LET    C = C - 7
30  POKE   16 388, C - 256 * INT (C/256)
40  POKE   16 389, INT (C/256)
50  NEW
```

Il ne vous reste plus qu'à entrer votre programme en langage machine en recalculant toutefois son adresse de début :

```
 10  LET C = PEEK 16 388 + 256 * PEEK 16 389
 60  POKE   C,237
 70  POKE   C + 1,75
 80  POKE   C + 2,54
 90  POKE   C + 3,64
100  POKE   C + 4,6
110  POKE   C + 5,0
120  POKE   C + 6,201
130  PLOT   25,30
140  PRINT  "L ABSCISSE EST:"; USR (C)
```

Ainsi avec cette méthode le sous-programme en langage machine est à l'abri d'une mauvaise commande : RUN, CLEAR, et même NEW.

Il existe par contre trois inconvénients :

— l'obligation d'exécuter la commande NEW pour pouvoir disposer effectivement de la place désirée en fin de mémoire,

— le fait de devoir réentrer le programme avant chaque première utilisation, celui-ci ne pouvant être sauvegardé sur cassette,

— l'adresse du début du sous-programme changera selon que vous utilisez une extension RAM ou non. Ceci vous obligera une fois de plus à ne travailler qu'avec des débranchements relatifs.

## 3.3. TABLEAU RÉCAPITULATIF

| | *Instruction REM* | *Tableau alphanumérique à une dimension* | *Modification de la variable RAMTOP* |
|---|---|---|---|
| Adresse du sous-programme | 16 514 (16 509 + 5) | PEEK 16 400 + PEEK 16 401 * 256 + 6 | PEEK 16 388 + PEEK 16 389 * 256 |
| Avantages | — Le sous-programme peut être sauvegardé avec le programme BASIC.<br>— Possibilités de détruire les lignes de création du programme. | — Le sous-programme peut être sauvegardé avec le programme BASIC.<br>— Possibilité de détruire les lignes ayant servi à la création du programme. | — Le sous-programme est à l'abri d'une mauvaise manipulation (commande RUN, CLEAR ou NEW). |
| Inconvénients | — Problèmes lors d'un listing du programme | — Éviter les commandes RUN et CLEAR.<br>— Le programme n'est pas implanté à une adresse fixe. | — Obligation d'exécuter une commande NEW pour disposer de la place mémoire.<br>— Le programme ne peut pas être sauvegardé sur cassette.<br>— Le programme n'est pas implanté à une adresse fixe. |

## 3.4. PROGRAMMES DE CHARGEMENT

Dans l'exemple choisi, le programme en langage machine est relativement court. Cependant pour les longs programmes vous pouvez trouver fastidieux de devoir écrire une longue suite d'instructions POKE. Le programme suivant vous permettra d'éviter ce pensum!

Ce programme vous demande quelle est la méthode que vous avez choisie pour écrire votre programme en langage machine.

RE pour l'utilisation de l'instruction REM.
TA pour l'utilisation du tableau.
RTOP pour la modification de la variable RAMTOP.



Suivant la méthode choisie vous aurez bien sûr auparavant, inclut une instruction REM, ou déclaré un tableau ou modifié la variable RAMTOP.

Vous n'aurez plus qu'à donner chaque fois que le ZX 81 vous le demandera le code hexadécimal correspondant à l'instruction désirée.

Pour arrêter le programme appuyer sur NEWLINE.

```
9 010  LET     RE = 16 514
9 020  LET     TA = 16 400
9 030  LET     RTOP = 16 388
9 040  PRINT   "METHODE CHOISIE: RE,TA OU RTOP?"
9 050  INPUT   M
9 060  GOTO    M - 7 320
9 065  REM     MODIFICATION DE RAMTOP
9 068  LET     M = PEEK (M) + PEEK (M + 1) * 256
9 070  GOTO    9 194
9 075  REM     TABLEAU ALPHANUMERIQUE
9 080  LET     M = PEEK (M) + PEEK (M + 1) * 256 + 6
9 100  REM     INSTRUCTION REM
9 194  PRINT   "CODE HEXA?"
9 200  INPUT   C$
9 205  CLS
9 210  IF      C$ " " THEN STOP
9 220  POKE    M, CODE C$ (1) * 16 + CODE C$ (2) - 476
9 230  LET     M = M + 1
9 240  GOTO    9 194
```

Ce petit programme peut-être amélioré: on pourrait par exemple revenir sur le dernier code entré en appuyant sur touche R. Il faudrait alors rajouter les instructions suivantes:

```
9 215  IF     C$ = " R " THEN GOTO 9 250
9 250  LET    M = M - 1
9 260  LET    V = PEEK M
9 270  PRINT  " ANCIEN CODE = ";
9 280  PRINT  CHR$(INT(V/16) + 28); CHR$(V - 16 * INT(V/16) + 28)
9 290  GOTO   9 194
```

On peut aussi utiliser la méthode suivante : Deux instructions REM au début du programme.

La première instruction REM servira à réserver la place en mémoire et la seconde à mettre le code hexadécimal. Cette dernière pourra être supprimée ensuite. Ainsi pour l'exemple que nous avons considéré nous aurons :

```
10 REM XXXXXXX
20 REM ED4B36400600C9
```

nous utiliserons alors le programme de chargement suivant :

```
CHARGEUR
9000 LET W=16511
9010 LET B=PEEK W+PEEK (W+1)*256
-2
9020 LET W=W+B+9
9030 FOR I=0 TO (B-1)*2 STEP 2
9040 LET A=PEEK (W+I)*16+PEEK (W
+I+1)-476
9050 POKE 16514+INT I/2,A
9060 NEXT I
9070 STOP
```

— l'instruction 9 010 permet de connaître la longueur du sous-programme écrit en langage machine,

— l'instruction 9 020 calcule l'adresse du début du code en hexadécimal, soit :

16 511 + 2 (pour le numéro de ligne) + 5 (pour les 5 premiers octets de l'instruction REM suivante : + 2 (pour REM et NEWLINE) + la longueur calculée B.

## 3.5. PASSAGE DE PARAMÈTRES

Il se peut que vous ayez besoin de passer des paramètres à votre sous-programme, ou que celui-ci doive vous en retourner. Plusieurs méthodes sont possibles.



### 3.5.1. Utilisation de la place mémoire inoccupée par le système

Cette méthode s'applique si vous avez peu de paramètres, 3 au maximum : vous pouvez alors utiliser les adresses 16 417, 16 507 et 16 508 qui font partie des variables système mais qui sont inutilisées par le ZX 81. Ces adresses ont l'extrême avantage d'être fixes. Elles seront chargées et lues par le programme BASIC en utilisant les instructions POKE et PEEK.

Si le nombre de paramètres que vous désirez passer est un peu plus élevé vous pouvez aussi, si vous n'avez pas d'imprimante, utiliser la place allouée au buffer d'imprimante : vous disposez alors de 32 octets ! L'adresse du début de cette zone est 16 444.

### 3.5.2. Réservation de place avec le programme

Utiliser le buffer de l'imprimante comme zone de stockage est avantageux, cependant le jour où vous ferez l'acquisition de celle-ci vous serez dans l'obligation de modifier vos programmes. C'est pourquoi il est préférable de réserver de la place derrière le programme lui-même, les valeurs des paramètres étant stockées par des instructions POKE avant l'appel du sous-programme.

**Remarques :**

**1.** Dans chaque programme écrit en langage machine il faudra être prudent si vous utilisez des branchements à des adresses absolues, certaines zones de la mémoire étant allouées dynamiquement (surtout si vous ne travaillez pas avec un bloc extension mémoire).

**2.** D'autre part il faudra veiller à ne pas détruire les registres secondaires A' et F', le registre d'index IX et le registre de rafraîchissement mémoire R du Z 80. Le registre d'index IY et le registre d'interruption I devront, s'ils sont utilisés, être respectivement réinitialisés à 4 000 H et 1E H (H = hexadécimal).

# 4

# Possibilités graphiques

Il y a encore quelques années, les ordinateurs étaient plutôt employés pour réaliser des calculs compliqués. Les résultats de ces calculs, généralement imprimés sous forme de tableaux peu clairs, étaient difficiles à exploiter pour les non initiés. Obtenir un résultat sous forme de graphe ou courbes nécessitait l'emploi de machines coûteuses et d'utilisation complexe.

Aujourd'hui les micro-ordinateurs semblent se rallier à cette petite phrase célèbre "un bon dessin vaut mieux qu'un long discours" et tous possèdent quelques instructions particulières qui vous permettent de faire des graphiques. Dans ce chapitre nous nous proposons donc d'étudier les possibilités graphiques de votre ZX 81.

## 4.1. SEMI-GRAPHIQUE

Dans ce paragraphe nous nous attacherons au fonctionnement des instructions PLOT et UNPLOT.

### 4.1.1. PLOT X, Y

**PLOT** x, y
x est l'abscisse,
y est l'ordonnée.

L'instruction PLOT permet de sélectionner et de noircir un point quelconque de l'écran désigné par ses coordonnées x et y. Dans ce cas, l'écran est adressé dans un système d'axes rectangulaires.

Les abscisses x peuvent prendre les 64 valeurs de 0 à 63.

Les ordonnées y peuvent prendre les 44 valeurs de 0 à 43.

L'origine est située dans le coin en bas à gauche de l'écran. Votre écran aura donc le format suivant (fig. 1).

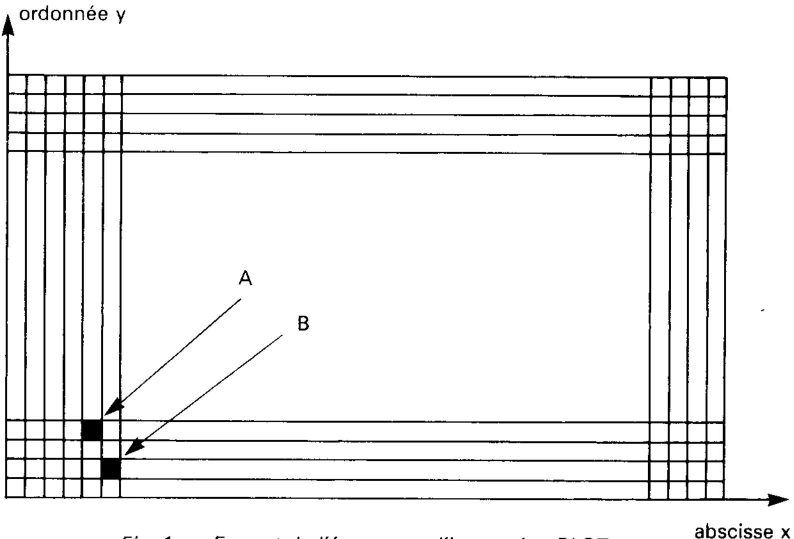



*Fig. 1 — Format de l'écran pour l'instruction PLOT*

Les points A et B de la figure 1 seront donc repérés par X = 4, Y = 3 pour le point A et par X = 5, Y = 1 pour le point B.

**Remarque :** Ce mode de désignation d'un point de l'écran est différent de celui utilisé dans l'impression de caractères qui se réfère lui à des coordonnées ligne, colonne telles que décrites plus loin.



*Exemple 1 :*

Un petit programme permettant de sélectionner ces points sera du type suivant :

```
10  REM    ESSAI DE PLOT
20  PLOT   4,3
30  PLOT   5,1
40  STOP
```

*Exemple 2 :*

Le programme suivant vous permettra de sélectionner n'importe quel point de l'écran.

```
 10  PRINT   "ESSAI DE PLOT"
 20  PRINT   "ENTREZ X"
 30  PRINT   "X DOIT ETRE COMPRIS ENTRE 0 ET 63"
 40  INPUT   X
 50  IF      X < 0 OR X > 63 THEN GOTO 20
 60  PRINT   "ENTREZ Y"
 70  PRINT   "Y DOIT ETRE COMPRIS ENTRE 0 ET 43"
 80  INPUT   Y
 90  IF      Y < 0 OR Y > 43 THEN GOTO 60
100  PLOT    X, Y
110  PRINT   "VOULEZ-VOUS CONTINUER ?"
120  PRINT   "REPONDEZ OUI OU NON"
130  INPUT   A$
140  IF      A$ = " NON " THEN STOP
150  CLS
160  GOTO    20
```

Pour sélectionner les quatre coins de l'écran il suffira de donner comme valeurs :

X = 0, Y = 0 pour le coin en bas à gauche
X = 63, Y = 0 pour le coin en bas à droite
X = 63, Y = 43 pour le coin en haut à droite
X = 0, Y = 43 pour le coin en haut à gauche

*Exemple 3 :*

Tracé des axes x et y d'un repère :

```
  3  FAST
  5  PRINT    "TRACE DES AXES X ET Y"
 10  REM      TRACE DE L'AXE DES X
 20  FOR      I = 0 TO 63
 30  PLOT     I, 0
 40  NEXT     I
 50  REM      TRACE DE L'AXE DES Y
 60  FOR      I = 0 TO 43
 70  PLOT     0, I
 80  NEXT     I
 90  SLOW
100  STOP
```

### 4.1.2. UNPLOT X, Y

UNPLOT x, y

x est l'abscisse (compris entre 0 et 63)
y est l'ordonnée (compris entre 0 et 43)

L'instruction UNPLOT permet d'annuler la sélection d'un point de l'écran. Ce point sera alors effacé.

*Exemple 1 :*

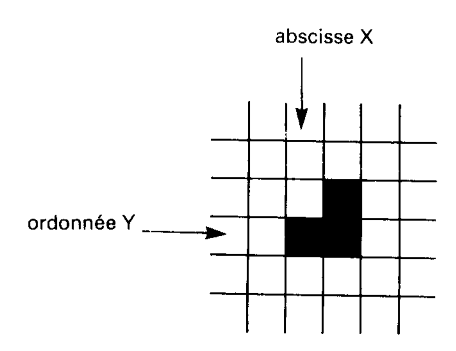
Reprenons l'exemple 2 de l'instruction PLOT et utilisons l'instruction UNPLOT au lieu de faire un effacement complet de l'écran comme cela était le cas avec la fonction CLS. Nous pouvons modifier la fin du programme de l'exemple 2 comme suit :

```
120  PRINT    "REPONDEZ OUI OU NON"
140  INPUT    A$
150  IF       A$ = " NON " THEN STOP
155  UNPLOT   X, Y
160  GOTO     20
```



## 4.2. CARACTÈRES GRAPHIQUES

L'instruction PLOT ne permet de sélectionner qu'un point à la fois. Supposons que l'on veuille représenter le schéma suivant :




Il faudrait utiliser trois instructions PLOT. Si X et Y sont les coordonnées du point en bas à gauche nous aurons le sous-programme suivant :

```
1 000  REM      SOUS-PROGRAMME DESSIN
1 010  PLOT     X, Y
1 020  PLOT     X + 1, Y
1 030  PLOT     X + 1, Y + 1
1 040  RETURN
```

Chaque fois que nous voudrons effectuer ce dessin nous inclurons dans le programme l'instruction XXXX GOSUB 1 000. Reconnaissez que ce serait extrêmement fastidieux !

Heureusement SINCLAIR a doté votre ZX 81 de plusieurs caractères graphiques qui vous permettront d'éviter ces solutions inélégantes. Vous trouverez la liste des caractères graphiques en annexe 1 de ce livre.

L'instruction PRINT AT permet d'en effectuer l'impression à un emplacement déterminé de l'écran au même titre que les autres caractères du clavier.

**PRINT AT** L, C

L est un numéro de ligne compris entre 0 et 21
C est un numéro de colonne compris entre 0 et 31

Dans le mode d'impression par caractère , l'emplacement de chaque impression est repéré dans un système d'axes rectangulaires " LIGNE-COLONNE ".

La clause AT de l'instruction PRINT permet de spécifier la ligne et la colonne où l'on désire afficher un caractère. Toutefois, contrairement à l'instruction PLOT, la ligne 0 est la première en haut de l'écran. D'autre part, si l'instruction PLOT autorise l'affichage de *64 points* par ligne et ce sur 44 lignes, l'instruction PRINT AT n'autorise que *32 caractères* par ligne sur 22 lignes. Ceci est dû au fait qu'un *caractère* a un encombrement double en hauteur et en largeur de celui d'un point.

Pour la fonction PRINT AT L, C; votre écran a donc le format suivant (fig. 2).

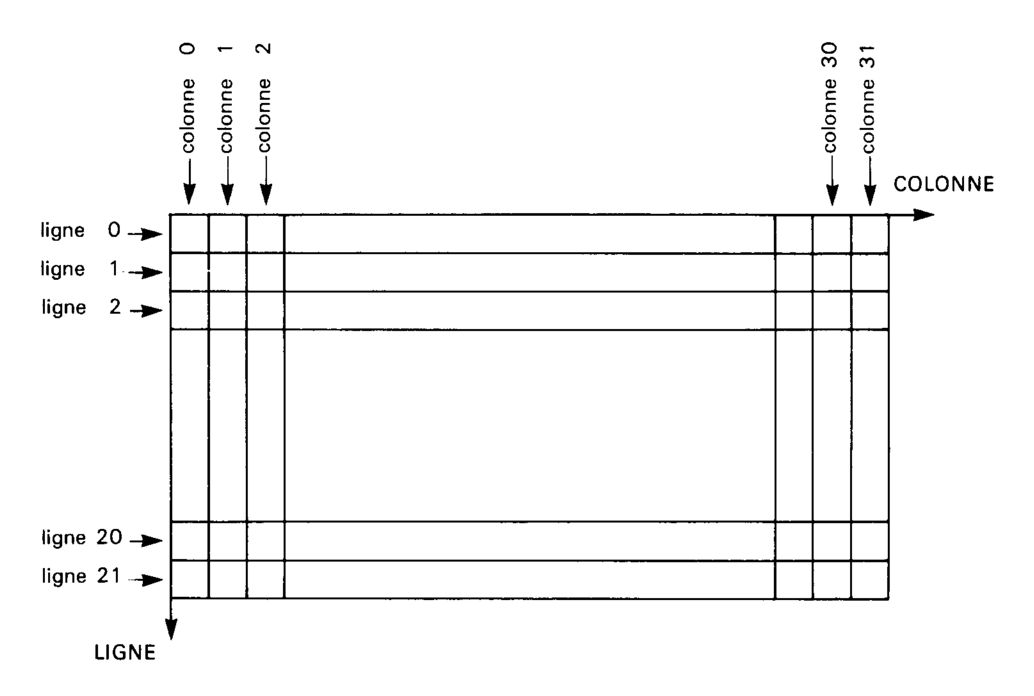



*Fig. 2 — Format de l'écran pour l'instruction PRINT AT*



*Exemple 1:*

Ainsi si nous désirons afficher le caractère graphique "■" (caractère espace en inversion vidéo) à chacun des quatre coins de l'écran nous utiliserons la séquence d'instructions suivantes:

```
10 REM    DELIMITE L ECRAN
20 PRINT  AT  0,0  ; "■"    caractère en haut à gauche
30 PRINT  AT  0,31 ; "■"    caractère en haut à droite
40 PRINT  AT 21,0  ; "■"    caractère en bas à gauche
50 PRINT  AT 21,31 ; "■"    caractère en bas à droite.
60 STOP
```

Voyez la différence avec PLOT. Les caractères sont quatre fois plus gros et les coordonnées différentes.

*Exemple 2:*

Comme pour l'instruction PLOT le programme suivant vous permet d'afficher n'importe quel caractère à l'endroit que vous spécifierez sur l'écran.

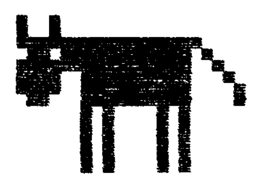
```
 10 PRINT  "ESSAI DE PRINT AT"
 20        CLS
 25 PRINT  " ENTRER L "
 30 PRINT  " L DOIT ETRE COMPRIS ENTRE 0 ET 21 "
 40 INPUT  L
 50 IF     L < 0 OR L > 21 THEN GOTO 20
 60 PRINT  " ENTRER C "
 70 PRINT  "C DOIT ETRE COMPRIS ENTRE 0 ET 31 "
 80 IF     C < 0 OR C > 31 THEN GOTO 60
 90 PRINT  " ENTRER VOTRE CARACTERE "
100 INPUT  C$
110 PRINT  AT L, C; C$
120 PRINT  " CONTINUEZ-VOUS? "
130 INPUT  R$
140 IF     R$ = " OUI " THEN GOTO 20
150 STOP
```

*Exemple 3 :*

Le programme suivant dessine un chien à l'aide des caractères graphiques

```
 10 REM     EXEMPLE DE GRAPHIQUE
 20 PRINT   AT 10,5; "▌ ▐"
 30 PRINT   AT 11,5; "▟ ▙ ■ ■ ■ ■ ■ ■ ▚"
 40 PRINT   AT 12,5; "■ ■ ▀ ■ ■ ■ ■ ■"; "  "; "▚"
 50 PRINT   AT 13,5; "▟ ▙"; "  "; "■ ■ ■ ■ ■"; "  "; "▌"
 60 REM     DESSIN DES PATTES
 70 FOR     I = 14 TO 16
 80 PRINT   AT I, 8; "▌ ▌"; "  "; "▐ ▐"
 90 NEXT    I
100 STOP
```

Vous obtiendrez le résultat suivant (copie du contenu de l'écran sur l'imprimante avec la commande COPY).



L'instruction PRINT AT peut aussi être utilisée avec la fonction CHR$.

*Exemple :* On désire dessiner un axe avec une flèche à son extrémité. Il faudra procéder de la façon suivante :

```
10  REM     AXE AVEC UNE FLECHE
20  FAST
30  FOR     I = 0 TO 30
40  PRINT   AT 10, I; CHR$ 22;
50  NEXT    I
60  PRINT   AT 10, 31; CHR$ 18
70  SLOW
80  STOP
```

**Remarque :** Un caractère imprimé aux coordonnées L, C occupera la place des quatre points ayant les coordonnées suivantes :

X1 = 2 * C,     Y1 = 2 * (21 — L)
X2 = 2 * C,     Y2 = 2 * (21 — L) + 1
X3 = 2 * C + 1, Y3 = 2 * (21 — L)
X4 = 2 * C + 1, Y4 = 2 * (21 — L) + 1



## 4.3. GRAPHIQUES ANIMÉS

Jusqu'à présent chaque fois que nous avons dessiné quelque chose, nous ne voyions apparaître le dessin que lorsque celui-ci était terminé. Ceci est dû au fait que nous demandions toujours l'exécution du dessin en mode FAST. Or dans ce mode le résultat ne sera affiché que si l'une des trois instructions suivantes doit être exécutée : PAUSE, STOP ou INPUT, ou si bien sûr votre programme est terminé. Pour le vérifier, essayez le programme suivant :

```
10  FAST
20  FOR     I = 63 TO 0 STEP — 1
30  PLOT    I, 40
40  NEXT    I
50  STOP
```

Dans ce premier cas vous voyez apparaître une ligne "complète" dans le haut de l'écran lorsque l'exécution du programme est terminée.

Remplacez l'instruction 10 FAST par la suivante : 10 SLOW et relancer l'exécution. Cette fois-ci vous voyez la ligne en haut de l'écran se " construire " au fur et à mesure. Ainsi pour faire du graphique animé il faut se mettre dans le mode SLOW.

Pour donner une impression de mouvement encore plus réelle il faudra effacer le point sur l'écran après l'avoir sélectionné. Le programme devient alors :

```
10  SLOW
20  FOR     I = 63 TO 0 STEP — 1
30  PLOT    I, 40
40  UNPLOT  I, 40
50  NEXT    I
60  STOP
```

Cette impression de mouvement peut aussi être obtenue avec l'instruction PRINT AT. Le caractère sera effacé en l'écrasant par un caractère blanc.

*Exemple:* Déplacement d'un navire

```
 10  REM      DEPLACEMENT D'UN NAVIRE
 20  SLOW
 30  REM      DESSIN DE LA MER
 40  FOR      I =  0 TO 31
 50  FOR      J = 10 TO 12
 60  PRINT    AT J,I; CHR$ 8
 70  NEXT     J
 80  NEXT     I
 90  REM      DEPLACEMENT DU NAVIRE
100  FOR      I = 25 TO 0 STEP − 1
110  GOSUB    200
120  NEXT     I
130  STOP
200  REM      DESSIN DU NAVIRE
210  PRINT    AT 8, I + 2; "▟ ▟ ▟ ▖"; " "
220  PRINT    AT 9, I; "▜ █ █ █ █ █"; " "
230  RETURN
```

Le caractère espace dans les deux instructions PRINT sert à effacer le caractère qui était affiché avant le déplacement. (i.e. ▖ et █).

Pour pouvoir intervenir sur le déroulement de ces graphiques comme c'est le cas pour de nombreux jeux, il faut utiliser la fonction INKEY$. (Pour plus de détails voir fonction INKEY$ dans le chapitre *LES INSTRUCTIONS BASIC DU ZX 81.*)

*Exemple:*

Avec le programme suivant nous faisons se poursuivre deux points sur l'écran.



```
 10  REM      POURSUITE
 20  FOR      I = 0 TO 53
 30  PLOT     I, 30
 40  PLOT     I + 10, 30
 50  UNPLOT   I, 30
 60  UNPLOT   I + 10, 30
 70  NEXT     I
 80  FOR      I = 53 TO 0 STEP − 1
 90  PLOT     I, 30
100  PLOT     I + 10, 30
110  UNPLOT   I, 30
120  UNPLOT   I + 10, 30
130  NEXT     I
140  GOTO     20
```

On désire que le premier des points fasse marche arrière lorsque l'on appuie sur la touche A. Il faudra alors rajouter les instructions suivantes:

```
 25  IF       INKEY$ = "A" THEN GOTO 150
 85  IF       INKEY$ = "A" THEN GOTO 150
150  FOR      J = 0 TO 4
160  PLOT     I + J, 30
170  PLOT     I + 10 − J, 30
180  UNPLOT   I + J, 30
190  UNPLOT   I + 10 − J, 30
200  NEXT     J
210  PRINT    "BOUM"
220  STOP
```

# 5
# Compatibilité ZX 80/ZX 81

## 5.1. COMPATIBILITÉ LOGICIELLE

Le ZX 80 a connu très rapidement un large succès et de nombreux programmes ont été écrits pour cette machine. Il serait dommage de ne plus pouvoir en profiter. Dans cette première partie nous énumérerons les différents points sur lesquels il faudra veiller pour pouvoir réutiliser ces programmes avec votre ZX 81.

### 5.1.1. Troncature

Le ZX 80 ne calcule que sur des nombres entiers alors que sur le ZX 81 on dispose d'une arithmétique en virgule flottante. Il faudra donc rétablir la troncature automatique qui est assurée avec le ZX 80. Pour cela utilisez la fonction INT.

On remarquera la différence suivante dans le cas où le nombre est négatif.

Avec le ZX 80 si le résultat exact est – 11.9 la troncature donne – 11.

Avec le ZX 81 en utilisant la fonction INT, – 11.9 donnera – 12.

### 5.1.2. Les boucles

Avec le ZX 81 si la valeur initiale de la variable de contrôle est supérieure à la valeur finale, les instructions qui composent le corps de la boucle ne seront pas exécutées, alors qu'avec le ZX 80 elles le sont au moins une fois...

### 5.1.3. Fonction TL$ (X$)

Cette fonction disponible sur le ZX 80 s'applique à une variable alphanumérique X$ ; elle permet de supprimer le premier caractère de la variable alphanumérique X$. Si cette variable est "vide" ou ne contient qu'un caractère, le résultat est une variable alphanumérique "vide". Cette fonction du ZX 80 peut être remplacée par la clause TO sur le ZX 81.

Ainsi :

```
10  LET  A$ = TL$ (A$)  sur le ZX 80 devra être écrit
10  LET  A$ = A$ (2 TO) sur le ZX 81.
```

### 5.1.4. Indice des tableaux

Sur le ZX 80 le premier élément d'un tableau est l'élément d'indice 0 alors que sur le ZX 81 c'est l'élément ayant l'indice 1.

### 5.1.5. Priorité des opérateurs

Les priorités des opérateurs * et / sont différentes sur le ZX 80 et sur le ZX 81. En effet, sur le ZX 80 la priorité de la multiplication est 8 et celle de la division 7 tandis que sur le ZX 81 ces priorités sont égales.

Une solution consiste à introduire des parenthèses afin de ne pas modifier l'ordre d'évaluation de l'expression.

Ainsi :

```
10  LET  A = B/C*D          sur le ZX 80 deviendra
10  LET  A = INT (B/(C*D))  sur le ZX 81.
```



### 5.1.6. Code des caractères

— Le ZX 80, et le ZX 81 n'utilisent pas toujours des codes identiques pour représenter le même caractère.

— Le tableau suivant établit une correspondance entre le ZX 80 et le ZX 81 pour ces différents codes.

| *Code ZX 80* | *Caractère sur le ZX 80* | *Code ZX 81* |
|---|---|---|
| 1 | " | 11 |
| 2 | ▌ | 5 |
| 3 | ▄ | 131 |
| 4 | ▘ | 1 |
| 5 | ▝ | 2 |
| 6 | ▖ | 4 |
| 7 | ▗ | 135 |
| 8 | ▚ | 6 |
| 9 | ▒ | 8 |
| 10 | 🮏 | 9 |
| 11 | 🮎 | 10 |
| 18 | − | 22 |
| 19 | + | 21 |
| 20 | * | 23 |
| 21 | / | 24 |
| 22 | = | 20 |
| 23 | > | 18 |
| 24 | < | 19 |

| *Code ZX 80* | *Caractère sur le ZX 80* | *Code ZX 81* |
|---|---|---|
| 129 | inversion vidéo de " | 139 |
| 130 | ▐ | 133 |
| 131 | ▀ | 3 |
| 132 | ▟ | 129 |
| 133 | ▙ | 130 |
| 134 | ▜ | 132 |
| 135 | ▛ | 7 |
| 136 | ▚ | 134 |
| 137 | ▒ | 136 |
| 138 | ▓ | 137 |
| 139 | ▓ | 138 |
| 146 | inversion vidéo de – | 150 |
| 147 | inversion vidéo de + | 149 |
| 148 | inversion vidéo de * | 151 |
| 149 | inversion vidéo de / | 152 |
| 150 | inversion vidéo de = | 148 |
| 151 | inversion vidéo de > | 146 |
| 152 | inversion vidéo de < | 147 |



| *Code ZX 80* | *Caractère sur le ZX 80* | *Code ZX 81* |
|---|---|---|
| 212 | " | 11 |
| 213 | THEN | 222 |
| 214 | TO | 223 |
| 215 | ; | 25 |
| 216 | , | 26 |
| 217 | ) | 17 |
| 218 | ( | 16 |
| 219 | NOT | 215 |
| 220 | – | 22 |
| 221 | + | 21 |
| 222 | * | 23 |
| 223 | / | 24 |
| 224 | AND | 218 |
| 225 | OR | 217 |
| 226 | ** | 216 |
| 227 | = | 20 |
| 228 | < | 19 |
| 229 | > | 18 |
| 230 | LIST | 240 |
| 231 | RETURN | 254 |
| 232 | CLS | 251 |
| 234 | SAVE | 248 |
| 237 | POKE | 244 |
| 239 | RANDOMISE | 249 |
| 240 | LET | 241 |
| 244 | PRINT | 245 |
| 246 | NEW | 230 |
| 248 | STOP | 227 |
| 249 | CONTINUE | 232 |
| 251 | GOSUB | 237 |
| 252 | LOAD | 239 |
| 254 | REM | 234 |

Les autres codes correspondent aux mêmes caractères ou ne sont pas utilisés par le ZX 80.

**Remarque:** Certains caractères du ZX 80 possèdent deux codes différents. Ceci est dû au fait que l'un des codes est celui généré lors d'un appui touche sur le clavier.

### 5.1.7. Tabulation

Dans le ZX 80 une ligne était partagée en quatre zones. Dans le ZX 81, nous ne disposons plus que de deux zones par ligne. Il faut alors utiliser la clause TAB avec l'instruction PRINT.

### 5.1.8. Variable système

C'est le point *le plus délicat* de cette partie. Le ZX 80 réserve 40 octets pour les variables système alors que le ZX 81 en réserve 125. Plusieurs de ces variables ont des adresses différentes. Dans le tableau ci-dessous nous établissons, *dans la mesure du possible,* une correspondance entre les adresses des variables système du ZX 80 et du ZX 81.

Cependant, nous conseillons à chacun de bien étudier ce qui est fait avec la variable système car une simple correspondance d'adresse peut ne pas toujours être suffisante.

| Adresse *décimale* sur le *ZX 80* | Nom de la variable sur le *ZX 81* | Longueur | Adresse *décimale* sur le *ZX 81* |
|---|---|---|---|
| 16 384 | ERR-NR | 1 | 16 384 |
| 16 385 | FLAGS | 1 | 16 385 |
| 16 386 | PPC | 2 | 16 391 |
| 16 388 | / | 2 | / |
| 16 390 | E-PPC | 2 | 16 394 |
| 16 392 | VARS | 2 | 16 400 |
| 16 394 | E-LINE | 2 | 16 404 |
| 16 396 | D-FILE | 2 | 16 396 |
| 16 398 | DF-CC | 2 | 16 398 |
| 16 400 | STKEND | 2 | 16 412 |
| 16 402 | DF-SZ | 2 | 16 418 |
| 16 403 | / | 2 | / |
| 16 405 | X-CTR | 2 | 16 408 |
| 16 407 | OLDPPC | 2 | 16 427 |
| 16 409 | / | 1 | / |
| 16 410 | T-ADDR | 2 | 16 432 |
| 16 412 | SEED | 2 | 16 434 |
| 16 414 | FRAMES | 2 | 16 436 |
| 16 416 | / | 2 | / |



| Adresse *décimale* sur le *ZX 80* | Nom de la variable sur le *ZX 81* | Longueur | Adresse *décimale* sur le *ZX 81* |
|---|---|---|---|
| 16 418 | / | 2 | / |
| 16 420 | S-POSN | 1 | 16 441 |
| 16 421 | / | 1 | 16 442 |
| 16 422 | / | 2 | / |

On veillera aussi à l'aspect suivant :

- *Sur le ZX 80 :* le buffer d'affichage se trouve après la zone VARIABLES et la zone mémoire de travail.
- *Sur le ZX 81 :* le buffer d'affichage se trouve avant la zone VARIABLES et la zone mémoire de travail.

— **Programme écrits en langage machine.**

- En utilisant une instruction REM, l'adresse de début est 16 427 sur le ZX 80 et 16 514 sur le ZX 81.
- En utilisant un tableau l'adresse de début est obtenue comme suit :

```
2 + PEEK (16 392) + 256 * PEEK (16 393) sur le ZX 80
6 + PEEK  16 400  + 256 * PEEK  16 401  sur le ZX 81
```

### 5.1.9. Fonction RND.

Sur le ZX 80 la fonction RND (n) fournit un nombre aléatoire compris entre 0 et n, alors que sur le ZX 81 elle donne un nombre compris entre 0 et 1 (1 étant exclu). Pour le ZX 81 cette fonction devra donc être introduite de la manière suivante :

```
INT (RND * N) + 1.
```

### 5.1.10 AND

Son évaluation est réalisée d'une façon différente sur le ZX 80.

Considérons l'instruction LET suivante :

```
10 LET A = ((B = 4)  AND  10)
```

*Sur le ZX 80* le résultat est A = 0 si B n'est pas égal à 4, et A = 10 si B est égal à 4. En effet, supposons que B soit égal à 4 l'expression est évaluée ainsi :

```
        10 LET A = ((4 = 4)  AND  10)
donc    10 LET A = (– 1      AND  10)
donc    10 LET A = 10
```

*Sur le ZX 81* si B est nul, le résultat est A = 0. Par contre, évaluons l'expression dans le cas où B est égal à 4.

```
        10 LET A = ((4 = 4)  AND  10)
donc    10 LET A = (1        AND  10)
donc    10 LET A = 1
```

Voir manuel SINCLAIR p. 71.

Il vaut donc mieux écrire pour le ZX 81 :

```
10 LET A = 10 AND B = 4.
```



### 5.1.11 Tableau récapitulatif

| N° du § | ZX 80 | ZX 81 | Ce qu'il faut faire |
|---|---|---|---|
| 1 | Troncature effectuée lors du calcul car BASIC ne travaille que sur des nombres entiers. | Troncature non effectuée car BASIC avec virgule flottante. | Utiliser la fonction INT. |
| 2 | Boucle parcourue au moins 1 fois si la valeur initiale supérieure à la valeur finale. | Boucle non effectuée si la valeur initiale supérieure à la valeur finale. | |
| 3 | Fonction TL$. | Pas de fonction TL$. | La remplacer par (2 TO). |
| 4 | Les indices des tableaux commencent à 0. | Les indices des tableaux commencent à 1. | Ajouter 1 à tous les indices. |
| 5 | Multiplication : priorité 8<br>Division : priorité 7 | Multiplication et division : priorité 8. | Introduire des parenthèses. |
| 6 | Certains caractères ont un code différent. | | Consulter le tableau au paragraphe 6. |
| 7 | Tabulation : 4 zones par ligne. | Tabulation : 2 zones par ligne. | Utiliser l'instruction PRINT avec TAB ou AT. |
| 8 | Variables systèmes situées à des adresses différentes. | | Consulter le tableau du paragraphe 8. |
| 9 | RND (N) : nombre aléatoire entre 1 et N. | RND : nombre aléatoire entre 0 et 1 (1 exclu) | Remplacer RND (N) par INT (RND * N) + + 1. |
| 10 | Fonction logique ET n'est pas réalisée de la même manière. | | |

## 5.2. COMPATIBILITÉ MATÉRIELLE

Vous avez un ZX 80 et vous enviez les heureux possesseurs de ZX 81 ! Ne vous tourmentez plus. Dans ce chapitre vous trouverez comment, au prix de quelques adaptations matérielles, transformer votre vieux ZX 80 de manière à le faire fonctionner comme un ZX 81, flambant neuf !...

### 5.2.1. Changement de la ROM

La version standard du ZX 80 possède une ROM BASIC de 4K. Toutefois la ROM BASIC de 8K peut être utilisée sur le ZX 80. Cette ROM offre les particularités suivantes :

- Arithmétique virgule flottante.
- Possibilités graphiques améliorées avec les instructions PLOT et UNPLOT.
- Chargement et sauvegarde de programmes en leur donnant un nom.
- Fonctions log, trigonométriques, et leurs fonctions inverses.
- Programmes interactifs avec la fonction PAUSE.
- BASIC prévu pour pouvoir faire fonctionner l'imprimante du ZX 81.
- Tableaux de caractères alphanumériques et tableaux numériques à plusieurs dimensions.

Pour placer cette ROM à l'intérieur de votre ZX 80, coupez d'abord l'alimentation ; ensuite, faites sauter les rivets en les perçant en leur centre avec une aiguille ou tout autre objet pointu et séparer les deux parties du coffret de votre ZX 80. La ROM 4 K est facilement identifiable. C'est le seul circuit intégré de la carte ayant 24 broches. Cette ROM s'enlève facilement, il suffit de la soulever à ses deux extrémités avec un petit tournevis. La ROM 8 K peut alors être introduite sur le support. Il suffit de veiller à ce qu'elle soit dans le bon sens, la broche numéro 1 (repérée par un point sur le boîtier) doit se trouver près du modulateur. Assurez-vous aussi que toutes les broches coïncident bien



avec les trous du support (si ce n'est pas le cas certaines risquent de se plier ou même de se casser !...). Il ne vous reste plus qu'à appuyer avec les pouces sur la ROM pour la faire rentrer dans son support. Vous pouvez alors remonter les deux parties du coffret de votre ZX 80.

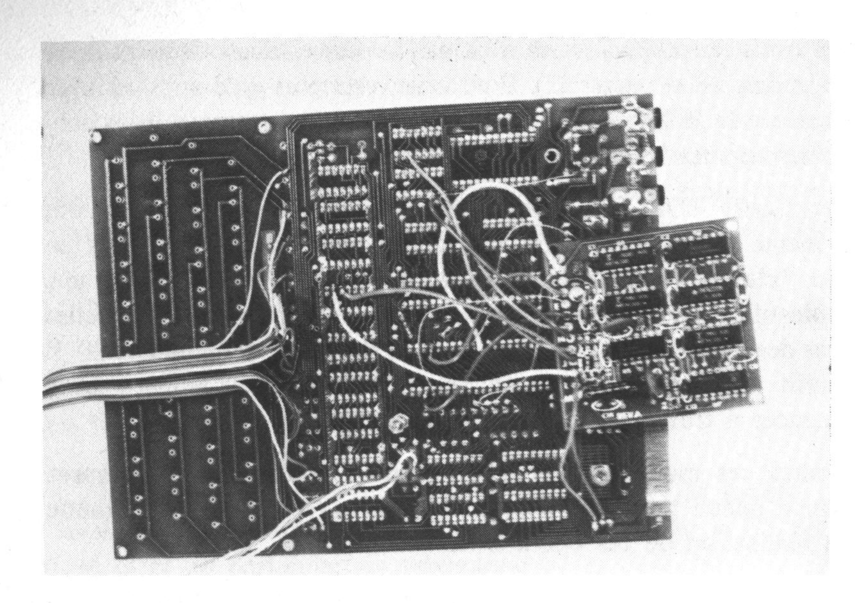
Avec cette ROM 8 K les touches du clavier de votre ZX 80 n'ont plus la même signification. SINCLAIR livre donc avec la ROM 8 K un nouveau "clavier" pour remplacer le précédent (c'est en réalité une feuille plastifiée sur laquelle est dessiné un clavier avec les nouvelles fonctions des touches). Ce clavier se pose au-dessus de celui du ZX 80. Il vous suffit maintenant de rebrancher l'alimentation et vous pouvez recommencer à travailler.

Toutes ces manipulations se réalisent aisément et vous n'aurez absolument aucun problème si vous faites preuve d'un peu de minutie dans la réalisation de ces opérations.

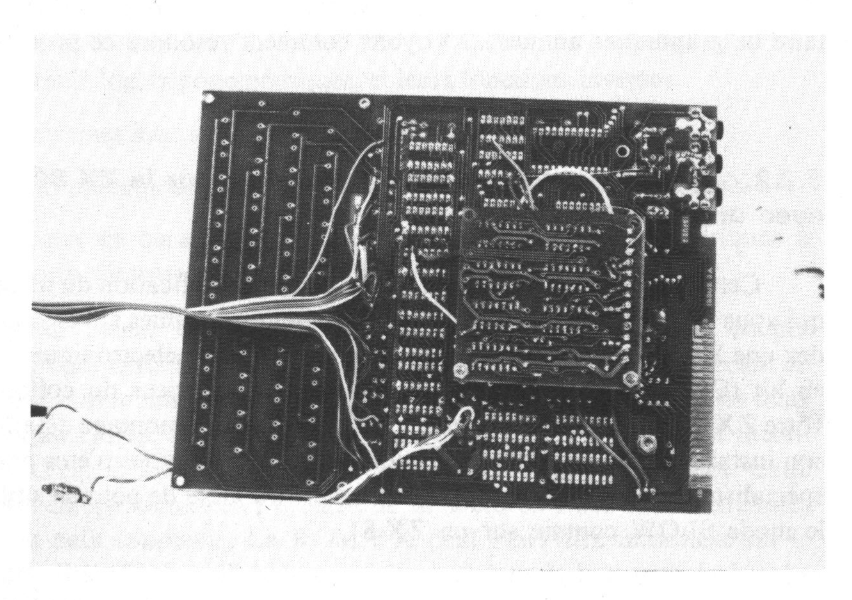
Vous possédez maintenant un ZX 80 qui fonctionne presque comme un ZX 81. En effet, vous disposez de l'arithmétique virgule flottante des instructions PLOT et UNPLOT, ... en résumé de toutes les facilités du ZX 81. Toutefois il existe un petit point noir à ce tableau. En effet vous ne disposez pas du mode SLOW ; vous ne pouvez donc pas faire de graphiques animés !... Voyons comment résoudre ce problème.

### 5.2.2. Adaptation de graphiques animés pour le ZX 80 avec une ROM BASIC 8 K

Cette adaptation consiste en réalité en une modification du matériel qui vous permettra alors de réaliser des graphiques animés si vous possédez une ROM BASIC 8 K. Il s'agit d'une petite carte électronique livrée en kit (COMSHOP LIMITED) qui se loge à l'intérieur du coffret de votre ZX 80. Cette carte est livrée avec un schéma de montage détaillé et son installation ne présente aucune difficulté même si vous n'êtes pas un spécialiste en électronique. Cela vous permettra alors de pouvoir utiliser le mode SLOW comme sur un ZX 81.



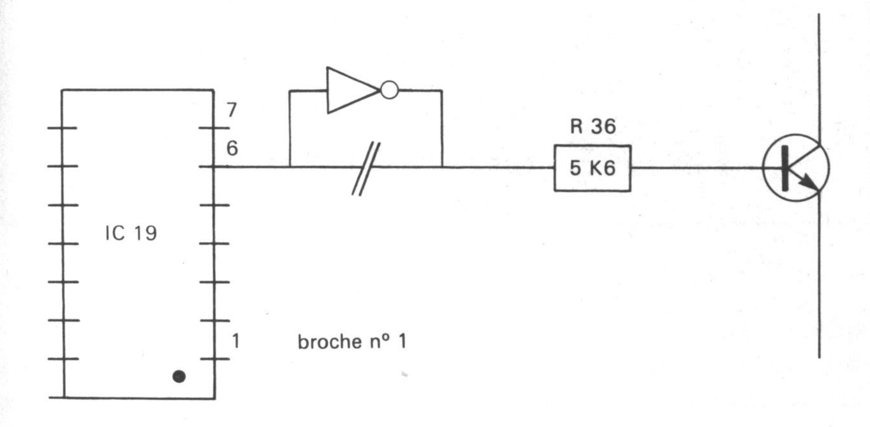
*Sur cette photo, on voit le dessous du circuit imprimé du ZX 80 et la carte d'adaptation avec les soudures.*



*Sur cette photo, on peut voir la carte d'adaptation montée sur le circuit imprimé du ZX 80.*



Toutefois cette carte étant fabriquée en Grande-Bretagne, il conviendra, pour la compatibilité avec le standard de télévision français, de couper la liaison entre IC 19 et R 36 et d'intercaler une porte inverseuse. (Voir schéma ci-dessous.)




*IC 19 et R 36 sont clairement répérés sur le circuit imprimé de votre ZX 80.*

Il vous faudra donc faire un peu d'électronique pratique !

# 6

# Extensions du ZX 81

Dans ce chapitre nous nous proposons de décrire quelques extensions du matériel. En effet s'il est relativement aisé d'obtenir des programmes pour le ZX 81 — de nombreuses sociétés en vendent sur cassettes — il est par contre plus difficile de trouver des cartes électroniques permettant d'augmenter ses possibilités.

## 6.1. AMÉLIORATION DE L'INTERFACE CASSETTE

Le ZX 81 possède déjà un interface cassette. Le montage proposé amplifie le signal de sortie du lecteur de cassette à un niveau de 3,5 V quand le ZX 81 effectue le chargement d'un programme. Ceci s'adresse surtout à ceux qui ont des problèmes de chargement (D BRUCE ELECTRONICS).

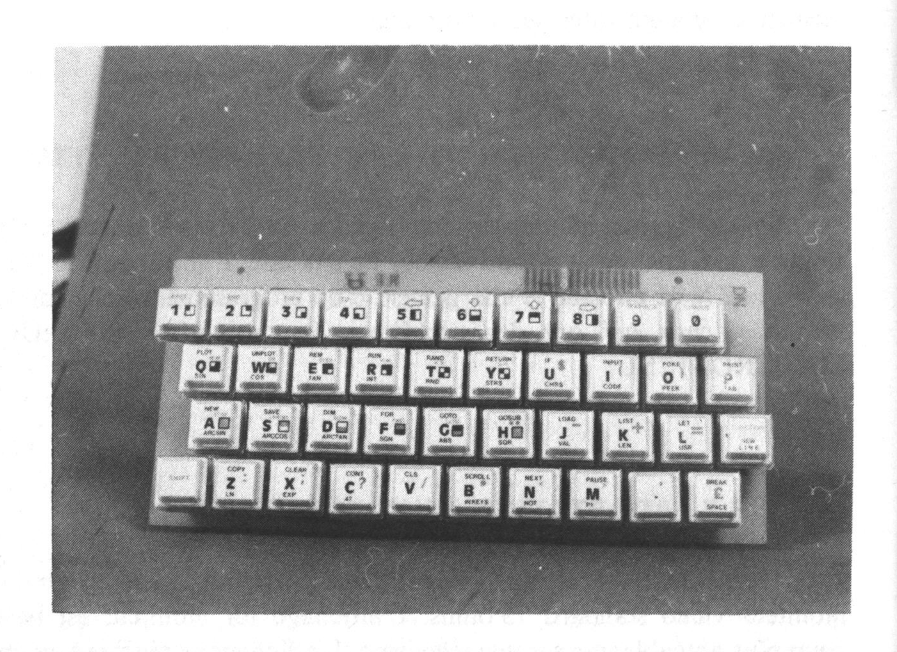
## 6.2. ADAPTATION D'UN MONITEUR VIDÉO

Ce montage fournit les signaux nécessaires pour la connexion d'un moniteur vidéo standard 75 ohms. L'affichage sur moniteur est beaucoup plus agréable que sur une télévision. L'affichage se réalisera en noir sur fond blanc (D BRUCE ELECTRONICS).

## 6.3. LES CLAVIERS

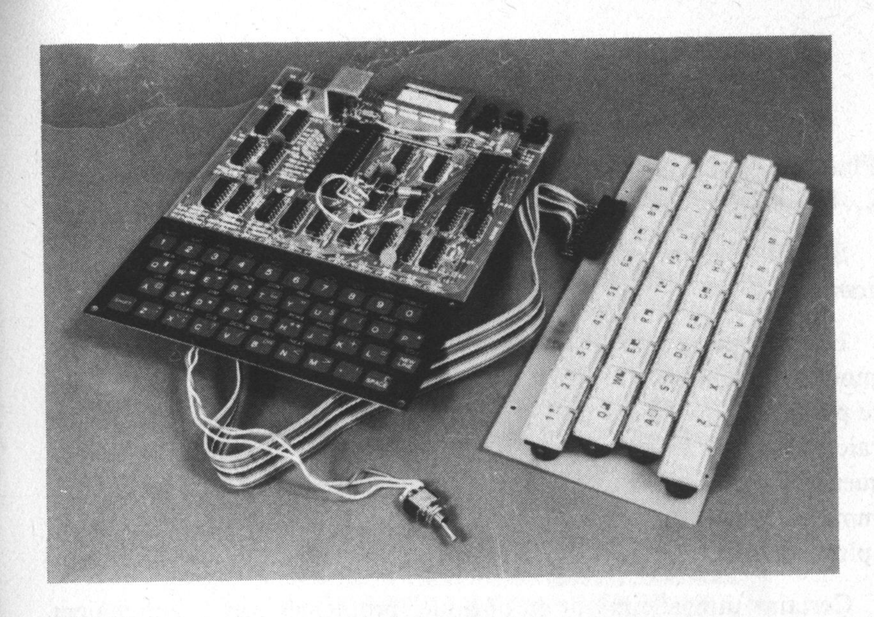
Le clavier du ZX 81 est assez peu commode à employer. En effet les touches du clavier ne sont pas en relief et l'enfoncement de l'une d'entre elles ne sera reconnue que si l'on exerce une pression suffisante. De ce fait on n'est jamais sûr que la touche ait bien été prise. Pour remédier à cela il existe un kit qui réalise un contrôle sonore de l'enfoncement des touches. Ce kit qui peut être monté à l'intérieur du ZX 81 émet un "bip" chaque fois qu'un appui touche est validé (D BRUCE ELECTRONICS).

D'autres constructeurs proposent d'ajouter un clavier traditionnel. Ces claviers se montent à la place de celui du ZX 81. Ces claviers ont l'avantage de permettre une frappe plus rapide ainsi qu'une diminution des erreurs. (REDDITCH ELECTRONICS, OK'TRONICS : ce dernier fabricant a prévu la place pour 40 touches supplémentaires qui seront utilisées pour pouvoir faire du graphique).

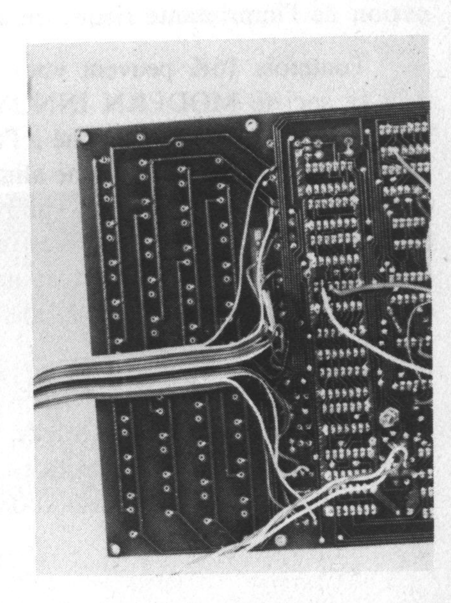


*Clavier du ZX 81.*





*Clavier monté, l'ancien clavier du ZX 80 est toujours utilisable.*



*Clavier monté sur un ZX 80. Il a fallu réaliser quelques soudures alors qu'avec un ZX 81 il s'enfiche directement dans un connecteur.*

## 6.4. LES EXTENSIONS MÉMOIRE RAM

Plusieurs sont disponibles sur le marché, leur taille varie de 2K à 48K (2K, 4K, 8K, 16K, 32K et 48K).

SINCLAIR propose une extension mémoire de 16K RAM *dynamique.*

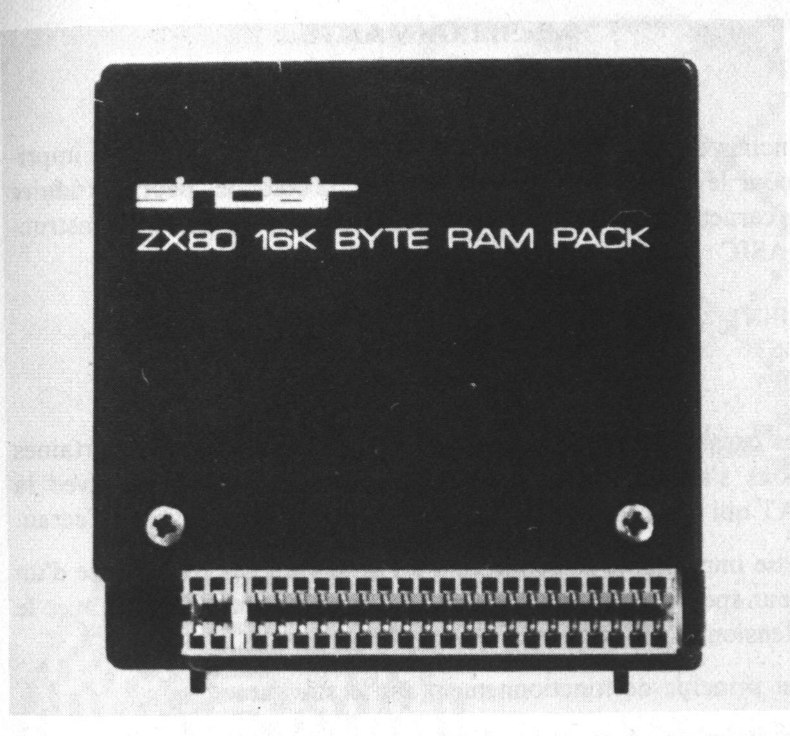
Le microprocesseur Z 80 permet de rafraîchir très simplement des mémoires dynamiques. Mais la conception astucieuse du ZX 81 utilise cette possibilité pour assurer d'autres tâches. Il faut donc des circuits de rafraîchissements indépendants, si l'on veut utiliser des mémoires dynamiques qui sont de loin les moins chères (et qui consomment le moins). Comme on le voit sur la photo la carte de rafraîchissement du bloc 16K est plus complexe que la carte mémoire elle-même !

Certains utilisateurs ont eu quelques problèmes car ils enfichaient et ils défichaient très souvent leur bloc-extension. Certaines personnes conseillaient de souder le bloc au connecteur du ZX 81 ! Nous vous conseillons plutôt de le laisser enfiché. En effet, si vous le soudez la connexion de l'imprimante risque de devenir délicate...

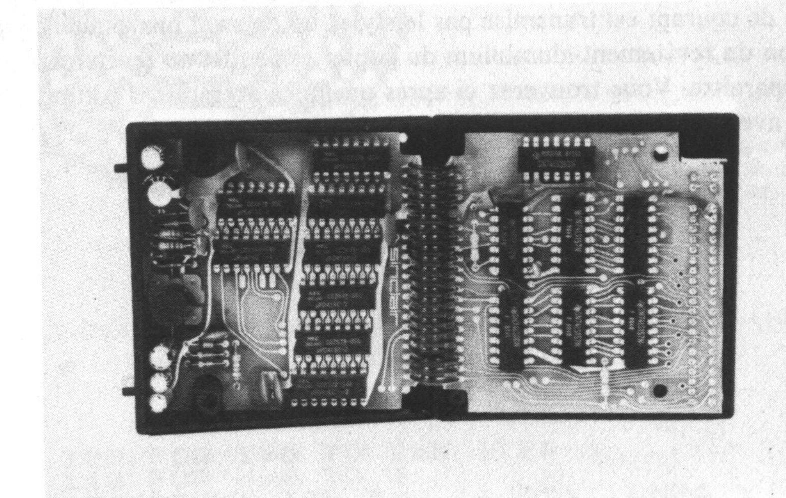
Toutefois 16K peuvent vous sembler insuffisants. Pour aller au delà la société MODERN INNOVATION fournit une carte de 32K RAM. Cette carte qui s'enfiche à l'arrière de votre ZX 81 comporte les circuits d'extension de bus, une alimentation et 32K de RAM... Les circuits d'extension du bus et l'alimentation permettront de connecter d'autres circuits. Cette carte n'est en réalité que le début d'une famille qui transformera votre ZX 81 en un système extrêmement complet. En effet ce constructeur prévoit la réalisation d'autres dispositifs comme :

— entrée-sorties parallèles,
— adaptation de Joy Sticks,
— entrée-sorties analogiques,
— graphique haute résolution.






*Le bloc extension 16K de SINCLAIR.*




*Bloc extension 16K: on distingue la carte mémoire (7 CI) et la carte rafraîchissement (8 CI).*

## 6.5. IMPRIMANTE

Sinclair est actuellement le seul fabricant qui propose une imprimante pour le ZX 81. Cette imprimante de 32 colonnes peut reproduire tous les caractères du ZX 81 et se commande à l'aide des trois instructions BASIC : LPRINT, LLIST et COPY.

LPRINT : impression du résultat
LLIST : sortie du listing sur l'imprimante
COPY : imprime ce qui est affiché sur l'écran

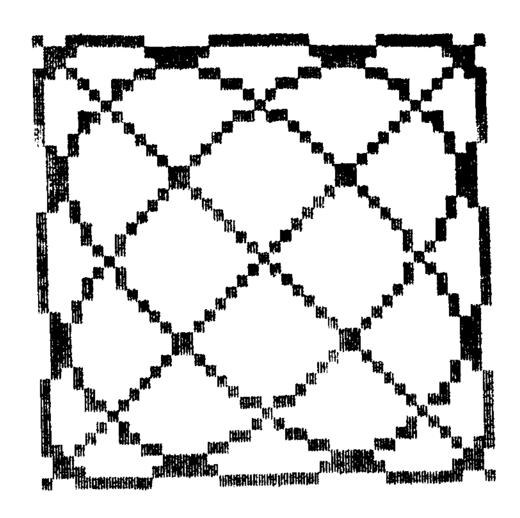
Les caractères à imprimer étant stockés dans un buffer, certaines restrictions s'appliquent à l'instruction LPRINT notamment avec la clause AT qui ne produira pas toujours le même résultat que sur l'écran.

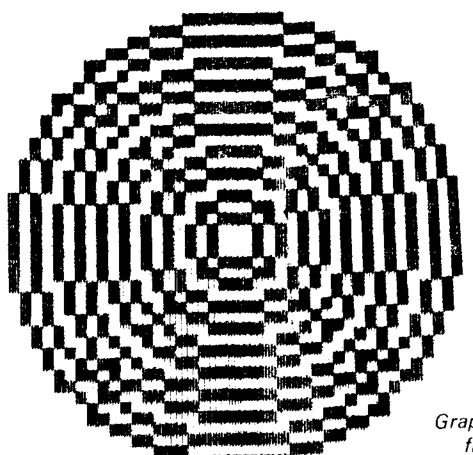
Cette imprimante se connectant à l'arrière du ZX 81, dispose d'un connecteur spécial pour vous permettre de continuer à travailler avec le bloc extension RAM.

Son principe de fonctionnement est le suivant :

Un stylet conducteur se déplace devant le papier revêtu d'une couche d'aluminium. Quand un point noir doit être imprimé, une impulsion de courant est transmise par le stylet, ce courant provoque l'évaporation du revêtement aluminium du papier et permet au revêtement noir d'apparaître. Vous trouverez ci-après quelques exemples d'édition obtenus avec l'imprimante.







*Graphiques obtenus à partir des programmes figurant dans le manuel d'utilisation de l'imprimante SINCLAIR du ZX 81.*

*Ci-dessous* — Exemple du jeu de caractères obtenu avec l'imprimante et le programme qui a été utilisé.

```
100 FOR I=0 TO 255 STEP 4
110 FOR J=0 TO 3
120 PRINT TAB 8*J; I+J; "="; CHR$
(I+J);
130 NEXT J
140 NEXT I
```

```
0=          1=          2=          3=
4=          5=          6=          7=
8=          9=          10=         11="
12=£        13=$        14=:        15=?
16=(        17=)        18=>        19=<
20==        21=+        22=-        23=*
24=/        25=;        26=,        27=.
28=0        29=1        30=2        31=3
32=4        33=5        34=6        35=7
36=8        37=9        38=A        39=B
40=C        41=D        42=E        43=F
44=G        45=H        46=I        47=J
48=K        49=L        50=M        51=N
52=O        53=P        54=Q        55=R
56=S        57=T        58=U        59=V
60=W        61=X        62=Y        63=Z
64=RND      65=INKEY$
                        66=PI       67=?
68=?        69=?        70=?        71=?
72=?        73=?        74=?        75=?
76=?        77=?        78=?        79=?
80=?        81=?        82=?        83=?
84=?        85=?        86=?        87=?
88=?        89=?        90=?        91=?
92=?        93=?        94=?        95=?
96=?        97=?        98=?        99=?
100=?       101=?       102=?       103=?
104=?       105=?       106=?       107=?
108=?       109=?       110=?       111=?
112=?       113=?       114=?       115=?
116=?       117=?       118=?       119=?
120=?       121=?       122=?       123=?
124=?       125=?       126=?       127=?
128=        129=        130=        131=
132=        133=        134=        135=
136=        137=        138=        139=
140=        141=        142=        143=
144=        145=        146=        147=
148=        149=        150=        151=
152=        153=        154=        155=
156=        157=        158=        159=
160=        161=        162=        163=
164=        165=        166=        167=
168=        169=        170=        171=
172=        173=        174=        175=
176=        177=        178=        179=
180=        181=        182=        183=
184=        185=        186=        187=
188=        189=        190=        191=
192="       193=AT      194=TAB     195=?
196=CODE
            197=VAL     198=LEN     199=SIN
200=COS     201=TAN     202=ASN     203=ACS
204=ATN     205=LN      206=EXP     207=INT
208=SQR     209=SGN     210=ABS     211=PEEK

212=USR     213=STR$
                        214=CHR$
                                    215=NOT
216=**      217= OR     218= AND
                                    219=<=
220=>=      221=<>      222= THEN
                                    223= TO
224= STEP
            225= LPRINT
```



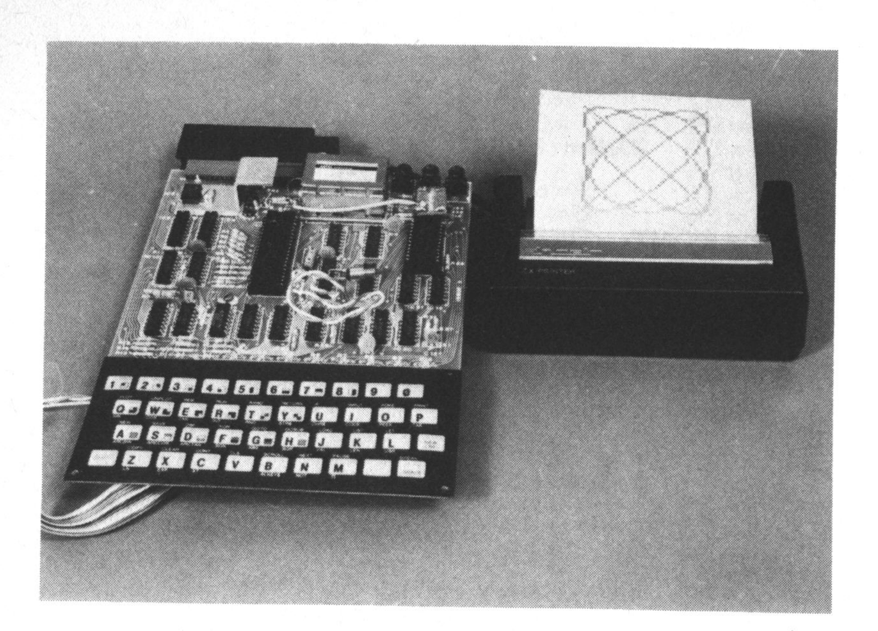
```
                      226= LLIST
                              227= STO
P
228= SLOW
        229= FAST
                230= NEW
                        231= SCR
OLL
232= CONT
        233= DIM
                234= REM
                        235= FOR

236= GOTO
        237= GOSUB
                238= INPUT
                        239= LOA
D
240= LIST
        241= LET
                242= PAUSE
                        243= NEX
T
244= POKE
        245= PRINT
                246= PLOT
                        247= RUN

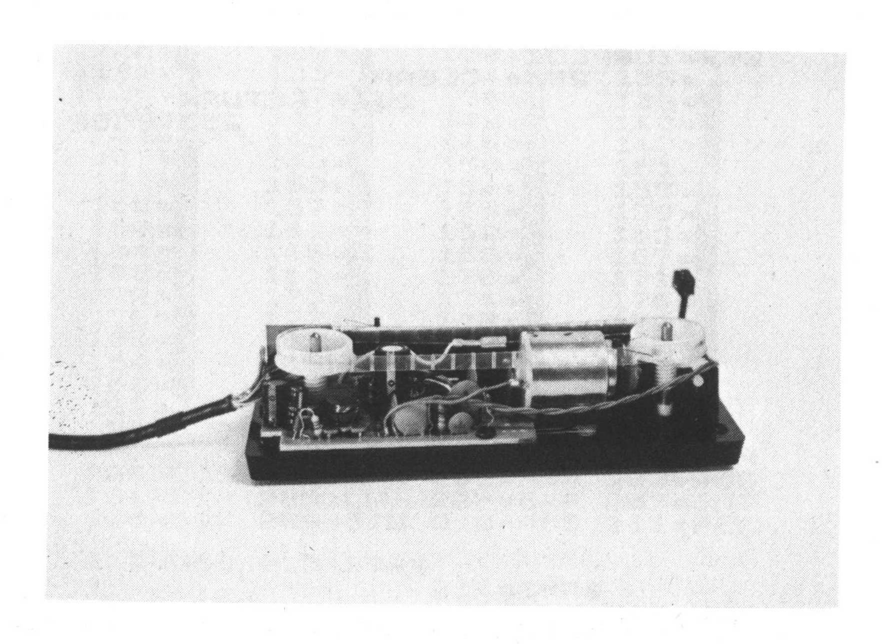
248= SAVE
        249= RAND
                250= IF  251= CLS

252= UNPLOT
        253= CLEAR
                254= RETURN
                        255= COP

Y
```



*Le ZX 81 et son imprimante.*



*L'imprimante du ZX 81 avec le capot enlevé.*



## 6.6. AUTRES EXTENSIONS

— **Port de 24 lignes entrée-sortie**

Ceci vous permettra de contrôler avec les interfaces nécessaires des LED, des relais, des générateurs de musique, etc... et de recevoir des signaux de différents capteurs, d'oscillateurs ou d'autres circuits intégrés. Cette carte a été conçue autour d'un circuit intégré MOS afin de réduire la consommation. Ce port est en "memory mapping" et pourra ainsi être programmé en utilisant les instructions PEEK et POKE. POKE pour la programmation du mode des ports entrée ou sortie et pour venir écrire sur les ports en sortie et PEEK pour lire le contenu des ports en entrée.

Le circuit intégré peut gérer trois ports (appelés ports A, B et C) de 8 bits chacun. Tous peuvent être commandés en entrée ou en sortie. De plus le port C pourra être séparé en deux avec quatre bits en entrée et quatre bits en sortie.

— **Affichage LED**

Ce circuit comporte 16 LED commandés par des "latches". Deux lignes de contrôle permettent de visualiser en continu des données ou d'éteindre les LED afin de diminuer la consommation en courant.

Ce montage peut être utilisé avec le précédent ainsi en se servant du port A comme port de contrôle il sera possible de visualiser les états des ports B et C.

## 6.7. BUS EXTENSION

Les différents circuits que nous venons de voir s'enfichent au dos du ZX 81 nous empêchant d'utiliser une extension mémoire. Pour remédier à cela le constructeur a prévu une carte d'adaptation qui permet l'utilisation d'une extension mémoire en même temps que deux autres circuits (port d'entrée-sortie et affichage par exemple...)

Cette carte comprend son propre régulateur 5 V afin de ne pas surcharger celui du ZX 81.

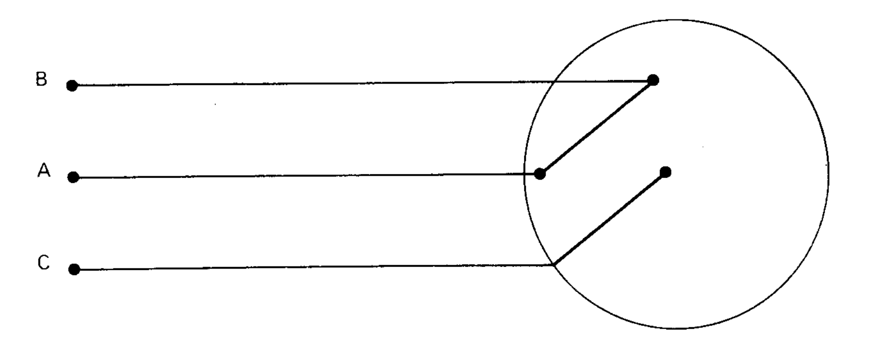
Sur le ZX 81 le décodage de la mémoire est un décodage incomplet. La carte d'adaptation a la place nécessaire pour l'implantation de deux décodeurs fournissant ainsi un décodage complet des adresses de tous les éléments du ZX 81 et des cartes connectées. Il est alors possible d'étendre la mémoire à 48K. Toutes ces cartes (port d'entrée-sortie, affichage, carte d'adaptation) sont au catalogue de la société REDDITCH ELECTRONICS.

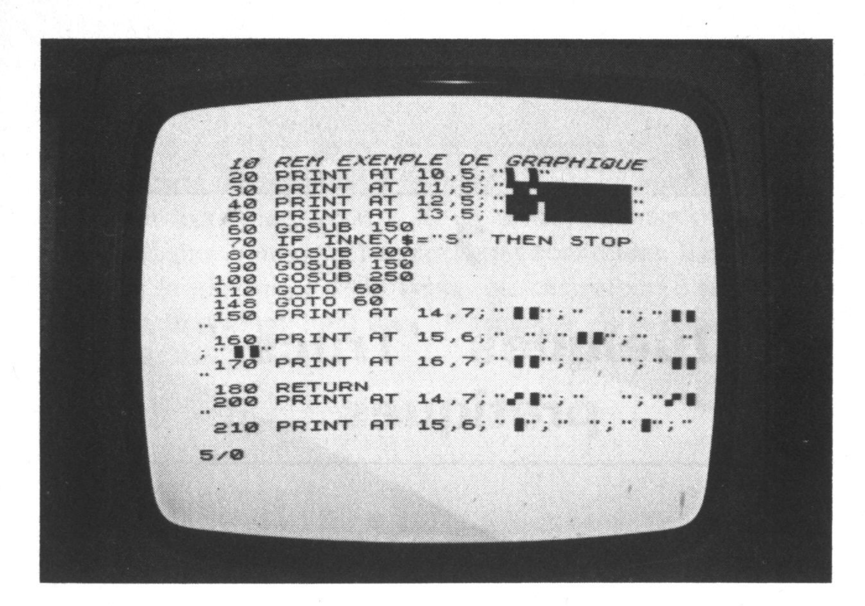


# 7
# Quelques "Trucs" pratiques

- **Faites-vous une inversion vidéo !**

Avec le montage suivant très simple : il suffit de rajouter un interrupteur inverseur, comme indiqué sur le schéma ci-dessous. Les repères A, B et C sont ceux qui figurent sur le circuit imprimé (du ZX 80).






*La vidéo comme elle apparaît habituellement sur votre ZX 80*

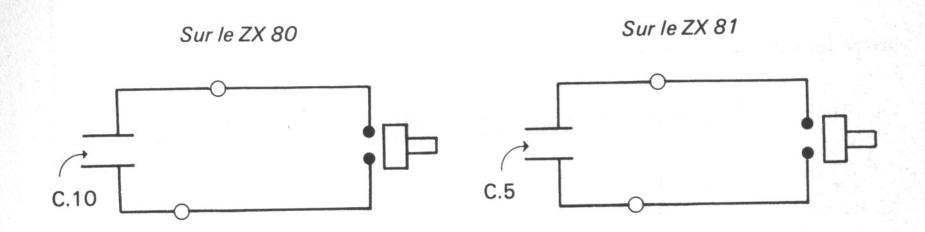



*La vidéo inversée*



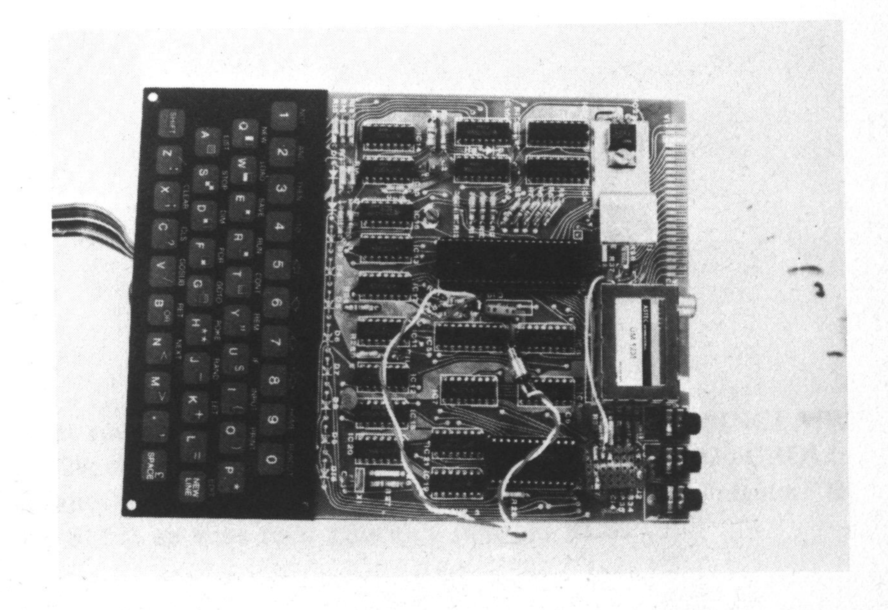
- **Mettez un RESET**

Pour vous éviter de couper l'alimentation de votre ZX 81 lorsque vous avez un problème il vaut mieux installer un bouton RESET. Le schéma est le suivant :




Ajoutez un interrupteur qui court circuite le condensateur.

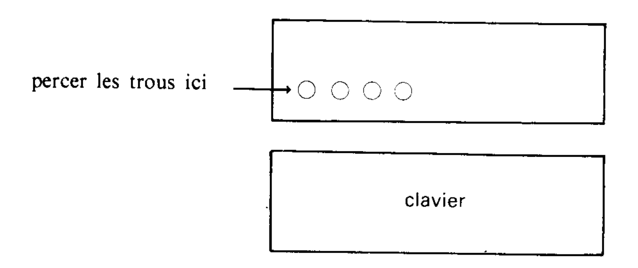
Toutefois, le contenu de la mémoire est quand même perdu.



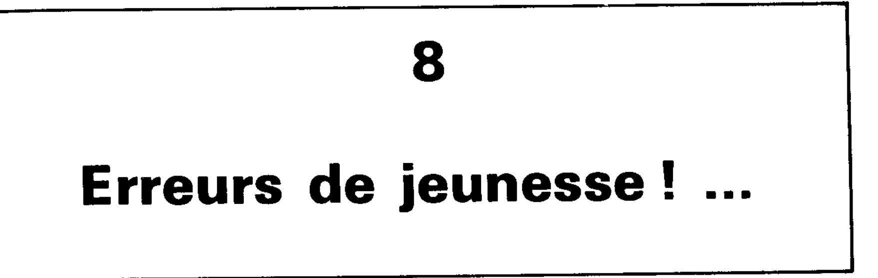
*Câblage du RESET et de l'INVERSION VIDÉO*

- **Aération**

Les circuits du ZX 81 chauffant quelque peu, une ventilation n'est pas inutile. Pour cela nous vous conseillons de percer quelques trous juste au dessous du sigle ZX 81. (3 ou 4 seront suffisants.)





# 8

# Erreurs de jeunesse ! ...

1. Essayez le programme suivant : *Normalement* le résultat devrait être :

```
      1 suivi          de 10 zéros
puis  2 "              "    "
       etc             .
                       .
                       .
                       .
puis 20 "              "    "
```

```
ERREUR DE CALCUL
  10 LET A=10**10
  20 FOR K=1 TO 20
  30 PRINT K*A
  40 NEXT K
```

2. Avez-vous essayé le programme donné dans le manuel SINCLAIR p. 129, chapitre 19 ? Rajoutez-lui l'instruction suivante : 42 SCROLL et faites-le exécuter en mode SLOW. Ce programme fait "planter" le ZX 81 ! Il ne vous reste plus qu'à faire un RESET.

3. Programmes SINCLAIR sur cassette.

— Le programme MÉTÉORITES comporte trois erreurs. Corrigez-les comme suit :

| | |
|---|---|
| ligne 220 | PRINT AT 14 – C * INT (C/4) * 4, RND * 30 ; "*"<br>remplacer – C * INT... par – C + INT... |
| ligne 270 | IF PEEK (P + K) = 23 THEN LET H = H + 1<br>il faut remplacer par IF... THEN LET H = H – 1 |
| ligne 300 | le dernier caractère graphique à imprimer doit être<br>▚ et non ▞ |

Ceci vous permettra d'avoir un dessin du vaisseau spatial symétrique.

⁂

— Le programme "jeu de la vie" fonctionne sur 1 K mais pas sur 16 K !



# 9

# Quelques programmes

## 1. Renumérotation

Ce programme permet d'effectuer une renumérotation d'un programme BASIC. Les GOTO et les GOSUB ne sont pas renumérotés.

```
8940 REM RENUMEROTATION
8945 REM SAUF GOTO ET GOSUB
8950 LET W=16509
8960 LET C=256
8990 PRINT "LIGNE DE DEBUT?"
9000 INPUT DE
9005 PRINT "LIGNE DE FIN?"
9010 INPUT FI
9015 PRINT "NUMERO DE DEBUT?"
9020 INPUT NL
9025 PRINT "INCREMENT?"
9030 INPUT PP
9040 GOSUB 9500
9050 IF A<DE THEN GOTO 9090
9060 IF A>FI THEN STOP
9070 POKE W,INT (NL/C)
9080 POKE W+1,NL-C*INT (NL/C)
9090 LET W=W+B
9095 LET NL=NL+PP
9100 GOTO 9040
9500 LET A=PEEK (W+1)+C*PEEK W
9510 LET B=4+PEEK (W+2)+C*PEEK (
W+3)
9520 RETURN
```

L'exécution du programme est lancée par la commande GOTO 8940.

LIGNE DÉBUT et LIGNE FIN sont les numéros de ligne de début et de fin avant renumérotation

NUMÉRO DE DÉBUT est le nouveau numéro de ligne que vous voulez donner.

INCREMENT est le nouveau pas de progression des numéros de ligne.

## 2. Décodage d'un programme

Ce programme affiche le contenu d'une partie de la zone PROGRAMME. Sur l'affichage chaque instruction sera séparée de la précédente par un saut de ligne.

```
8990 REM DECODAGE D""UN PROGRAMM
E
9000 LET C=256
9010 LET W=16509
9020 PRINT "NUMERO DE LIGNE DE D
EBUT?"
9030 INPUT D
9040 PRINT "NUMERO DE LIGNE DE F
IN?"
9050 INPUT F
9060 GOSUB 9500
9070 IF A<D THEN GOTO 9145
9080 IF A>F THEN STOP
9090 FOR I=0 TO B-1
9100 PRINT PEEK (W+I);" ";
9110 NEXT I
9130 PRINT
9140 PRINT
9145 LET W=W+B
9150 GOTO 9060
9500 LET A=PEEK (W+1)+C*PEEK W
9510 LET B=4+PEEK (W+2)+C*PEEK (
W+3)
9520 RETURN
```

L'exécution du programme est lancée par la commande GOTO 8990.



## 3. Affichage du contenu de la zone VARIABLE ou du Buffer d'affichage

```
9 000  PRINT  "VARIABLES OU BUFFER AFFICHAGE (V OU
              BF)"
9 010  INPUT  T$
9 020  PRINT  "NOMBRE D OCTETS?"
9 030  INPUT  N
9 040  DIM    A (N)
9 050  IF     T$ = "V" THEN LET I = PEEK 16 400 + PEEK
              16 401 * 256
9 060  IF     T$ = "BF" THEN LET I = PEEK 16 396 + PEEK
              16 397 * 256
9 070  FOR    K = 0 TO N - 1
9 080  LET    A (K + 1) = PEEK (I + K)
9 090  NEXT   K
9 100  FOR    I = 1 TO N
9 110  PRINT  A (I); " ";
9 120  NEXT   I
9 130  STOP
```

L'exécution est lancée par la commande GOTO 9 000.

**4. Jeu des allumettes** (avec extension 16 K). Programme paru dans la revue MICROSYSTEMES numéro 6 (JUILLET/AOUT 1979) et adapté au ZX 81.

Ce programme, lorsque vous jouez pour les premières fois avec lui est "ignorant". D'ailleurs vous gagnerez toujours. Cependant, au fur et à mesure que vous jouerez, il deviendra plus "intelligent" chaque fois, pour devenir finalement imbattable!

```
   1 DIM P(40,3)
   2 DIM M(40,40)
  10 REM JEU DES ALLUMETTES
  20 REM
  30 PRINT AT 10,7;"JEU DES ALLU
METTES"
  40 PAUSE 100
  50 CLS
  60 PRINT "AU DEPART,ON A 30 AL
LUMETTES."
  70 PRINT "ON DOIT EN RETIRER A
 TOUR DE"
  80 PRINT "ROLE .CELUI QUI RETI
RE LA       DERNIERE A PERDU."
  90 PRINT
 100 PRINT "CHACUN PEUT EN RETIR
ER UNE      QUANTITE QUI VARIE E
NTRE 1 ET LE DOUBLE DE CE QUE LE
 JOUEUR     PRECEDENT A PRIS."
 130 PRINT
 140 PRINT "AU PREMIER TOUR,ON P
EUT EN      PRENDRE SOIT 1 SOIT
2."
 150 LET N=30
 160 LET N1=N
 170 IF INKEY$="" THEN GOTO 170
 175 GOTO 300
 180 PRINT "ON RECOMMENCE? ";
 185 INPUT A$
 190 PRINT A$
 200 IF A$(1)="N" THEN STOP
 210 PRINT
 220 PRINT "MEME NOMBRE D""ALLUM
ETTES? ";
 230 INPUT A$
 235 PRINT A$
 240 IF A$(1)="O" THEN GOTO 295
 250 PRINT "ON EN MET COMBIEN? "
;
 260 INPUT N1
 265 PRINT N1
 270 IF N1<6 OR N1<>INT N1 THEN
GOTO 250
 280 IF N1<41 THEN GOTO 295
 285 PRINT "PAS PLUS DE 40 S.V.P
."
 290 GOTO 250
 295 PAUSE 70
 300 CLS
 310 LET N=N1
 320 LET N3=1
 330 LET N2=2
 340 FOR I=1 TO 40
 350 LET P(I,1)=0
 360 LET P(I,2)=0
 370 LET P(I,3)=0
 380 NEXT I
 390 PRINT "VOULEZ-VOUS COMMENCE
R? ";
 400 INPUT A$
 410 PRINT A$
 420 IF A$(1)<>"N" THEN GOTO 500
 430 IF A$<>"NUL" THEN GOTO 690
```



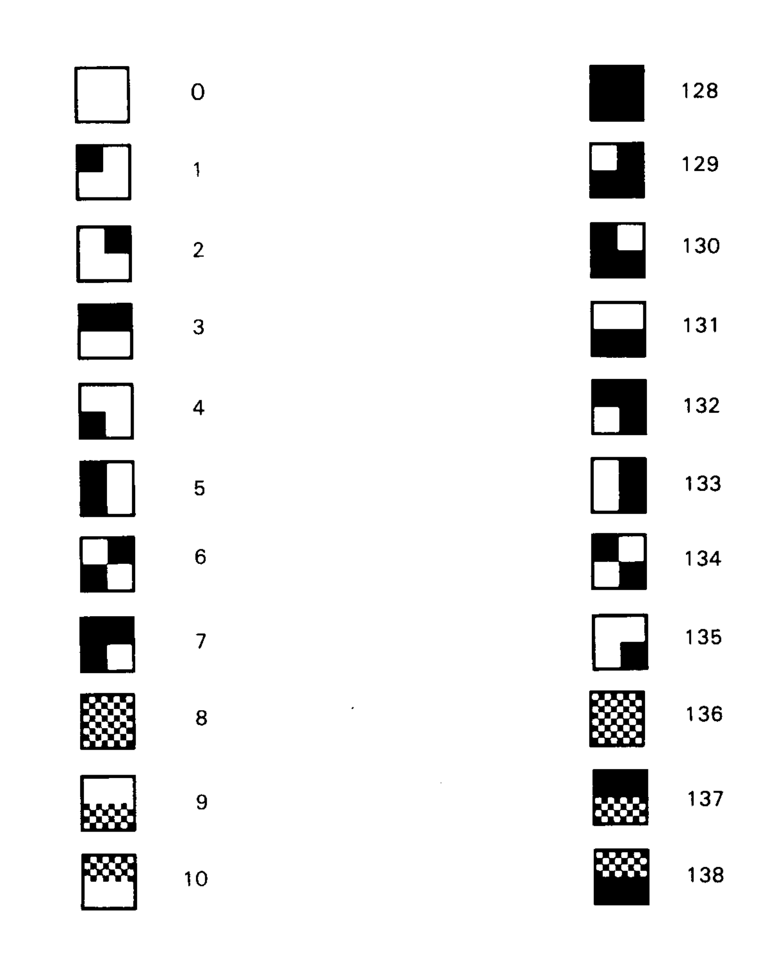
```
 440 FAST
 450 FOR I=1 TO 40
 460 FOR J=1 TO 40
 470 LET M(I,J)=0
 480 NEXT J
 490 NEXT I
 495 SLOW
 500 PRINT "COMBIEN EN PRENEZ-VO
US? ";
 510 INPUT C
 520 PRINT C
 530 IF C<1 OR C<>INT C THEN GOT
O 500
 540 IF C<=N2 THEN GOTO 570
 550 PRINT "ENTRE 1 ET ";N2;" S.
V.P."
 560 GOTO 500
 570 IF C<=N THEN GOTO 600
 580 PRINT "MAIS IL N""EN RESTE
QUE ";N;" ???"
 590 GOTO 500
 600 IF C<>N THEN GOTO 630
 605 PRINT
 610 PRINT TAB 7;"***J""AI GAGNE
***"
 615 LET D=1
 620 GOTO 710
 630 LET P(N3,1)=N
 640 LET P(N3,2)=N-C
 650 LET P(N3,3)=-1
 660 LET N=N-C
 670 LET N3=N3+1
 680 LET N2=2*C
 690 IF N<>1 THEN GOTO 800
 695 PRINT
 700 PRINT TAB 4;"BON,D""ACCORD.
J""AI PERDU."
 705 LET D=-1
 710 FOR I=1 TO N3-1
 715 LET N=P(I,1)
 720 LET N2=P(I,2)
 730 LET C=D*P(I,3)
 740 LET M(N,N2)=M(N,N2)+C
 750 NEXT I
 760 PRINT
 770 GOTO 180
 800 LET M1=-1000
 810 IF N2>N-1 THEN LET N2=N-1
 820 FOR I=N2 TO 1 STEP -1
 830 IF M(N,N-I)<=M1 THEN GOTO 8
60
 840 LET M1=M(N,N-I)
 850 LET C=I
 860 NEXT I
 870 PRINT "J""EN PRENDS ";C;
 880 LET P(N3,1)=N
 890 LET P(N3,2)=N-C
 900 LET P(N3,3)=1
 905 LET N=N-C
 910 LET N3=N3+1
 920 LET N2=2*C
 930 PRINT ".IL EN RESTE ";N
 940 IF N=1 THEN GOTO 605
 950 GOTO 500
```

La toute première fois que vous jouez, lancez l'exécution du programme par RUN. Lorsque vous déciderez de vous arrêter, faites une sauvegarde sur cassette. Lorsque vous désirez rejouer, rechargez le programme et lancez son exécution par la commande GOTO 30.



# ANNEXE 1

# Les caractères graphiques du ZX 81 et leurs codes



# ANNEXE 2

# Les codes d'erreur du ZX 81

| | |
|---|---|
| 0 | Terminaison normale de l'exécution |
| 1 | La variable de contrôle d'une boucle n'a pas été initialisée par une instruction FOR et il existe une variable ayant ce nom |
| 2 | Utilisation d'une variable indéfinie |
| 3 | Indice supérieur à la dimension du tableau |
| 4 | Manque de mémoire disponible |
| 5 | Les 21 lignes d'affichage sur l'écran sont remplies. Il faut exécuter une commande CONT |
| 6 | Dépassement de capacité. Le nombre est supérieur à $10^{38}$ |
| 7 | Une instruction RETURN a été reconnue sans l'exécution préalable de l'instruction GOSUB |
| 8 | Utilisation de l'instruction INPUT comme commande |
| 9 | Exécution d'une instruction STOP |
| A | Argument de fonction invalide |
| B | Le nombre entier obtenu après arrondi est en dehors des limites attendues |

C La variable alphanumérique d'une fonction VAL n'est pas une expression numérique

D Arrêt du programme par BREAK ou par STOP en réponse à une instruction INPUT

E Inutilisé

F Commande SAVE sans nom de programme



# ANNEXE 3

# Adresses utiles

— D. BRUCE ELECTRONICS
THE BEACON, BLACHALL ROCKS
CLEVELAND TS 274 BH
GRANDE BRETAGNE
TÉL. PETERLÉE (0783) 86.36.12

— BUG BYTE
MICROCOMPUTER SOFTWARE
98 — 100 THE ALBANY
OLD MALL STREET
LIVERPOOL L39EP
GRANDE BRETAGNE

— COMSHOP LIMITED
14 STATION ROAD
NEW BARNET
HARTSORDSHIRE EN51QW
GRANDE BRETAGNE

— MODERN INNOVATIONS
5 BRENCHLEY CLOSE
ROCHESTER KENT
GRANDE BRETAGNE

— OK TRONICS
23 SUSSEX ROAD
GORLESTON-ON-SEA
GREAT YARMOUTH — NORFLOK
GRANDE BRETAGNE
TÉL. (0493) 60.24.53

— REDDITCH ELECTRONICS
21 FERNEY HILL AVENUE
WORCESTERSHIRE 3974 RU
GRANDE BRETAGNE
TÉL. (0527) 61240.



Imprimerie de la Manutention à Mayenne
Dépôt légal : novembre 1982
N° d'Édition : 3855

Avec LA CONDUITE DU ZX81 obtenez le maximum de votre micro-ordinateur SINCLAIR ZX81. Cette lecture vitale vous apprendra comment réaliser des programmes en langage machine, économiser la place mémoire, chaîner des programmes sur cassette avec passage de paramètres, faire des graphiques animés ! ... Vous possédez un ZX80 ? Sachez alors ce qu'il faut faire pour utiliser la commande SLOW comme sur le ZX81 ! Vous trouverez de plus des conseils sur la manière d'adapter les programmes du ZX80 au ZX81. Saviez-vous qu'il existe des extensions augmentant encore le niveau de sophistication et de performance de votre ZX81 ? Découvrez-les avec LA CONDUITE DU ZX81.




8598